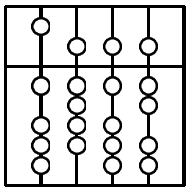
FAKULTÄT FÜR INFORMATIK
TECHNISCHE UNIVERSITÄT MÜNCHEN

Diplomarbeit in Informatik

Realisierung Virtueller Organisationen als Grid-Ressourcen

Natalia Cojocaru



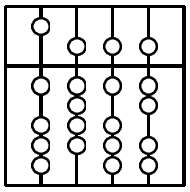
FAKULTÄT FÜR INFORMATIK
TECHNISCHE UNIVERSITÄT MÜNCHEN

Diplomarbeit in Informatik

Realisierung Virtueller Organisationen als Grid-Ressourcen

Bearbeiterin: Natalia Cojocaru
Aufgabensteller: Prof. Dr. Heinz-Gerd Hegering
Betreuer: Michael Schiffers

Abgabedatum: 15. September 2007



Ich versichere, dass ich diese Diplomarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 15. September 2007

. . . . . .

(Unterschrift des Kandidaten)

Trotz der Leistungsfähigkeit heutiger Rechner werden die meisten überwiegend kaum genutzt. Einzelne Peaks wechseln sich mit langen Untätigkeitsphasen ab. Auf der anderen Seite gibt es aber durchaus Projekte, deren Anforderung an Rechenleistung nicht abgedeckt wird. Außerdem laufen solche Projekte auf begrenzte Zeit, so dass eine Anschaffung von ausreichender Hardware sich nicht amortisieren würde.

In einem Grid [FoKe 04] wird die gemeinsame Nutzung von Grid-Ressourcen wie teuere Hochleistungsrechner, Software, Rechnernetzwerke, Daten und Sensoren ermöglicht. Die Rechenleistung wird dadurch zu einem Produkt, das auf Abruf bereit steht, was Grids auch die Bezeichnung „Rechenleistung aus der Steckdose“ eingebracht hat.

Da ein Grid potentiell wesentlich mehr Ressourcen beinhaltet, als im Einzelnen benötigt, ermöglichen Virtuelle Organisationen (VO) die Gruppierung der Untermenge von Grid-Nutzern und Ressourcen. Die Implementierung solcher Organisationen ist als notwendig anerkannt, im Detail aber nicht geklärt. Auch ist die Frage offen, in welcher Form und mittels welcher Schnittstellen die Virtuellen Organisationen auf einfache und effiziente Weise verwaltet werden können. An diesem Punkt werden wir anknüpfen.

Zu Beginn dieser Arbeit werden die Grundkonzepte und -begriffe, wie Web- und Grid-Services, Globus Toolkit 4, das Web Services Resource Framework (WSRF), Virtuelle Organisationen und andere ausführlich dargestellt.

Im Weiteren verknüpfen wir die erläuterten Themen und entwickeln daraus einen Ansatz, mit dem Virtuelle Organisationen verwaltet werden können. Dazu gehört die Definition von entsprechenden Web Services und dazugehörigen Ressourcen, sowie auch generelle Konzeption. Anschließend werden wir die gewonnenen Erkenntnisse zusammen mit den vorher definierten Komponenten innerhalb eines genau abgegrenzten Rahmens in die Praxis umsetzen. Ein einfaches Verwaltungstool wird die Funktionsfähigkeit und Korrektheit des Ansatzes beweisen.

Ziel dieser Diplomarbeit ist es, Beschreibungsmöglichkeiten Virtueller Organisationen als Ressourcen im Rahmen von Web Services Resource Framework festzulegen bzw. zu analysieren. Auf Basis dieser Erkenntnisse wird anschließend das VO-Konstrukt im Kontext des Globus Toolkits 4 implementiert.

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Motivation und Zielsetzung . . . 1
- 1.2 Fragestellung . . . 4
- 1.3 Vorgehensmodell . . . 5

2 Begriffliche Grundlagen **8**
- 2.1 Web Services . . . 8
- 2.2 Web Services Resource Framework (WSRF) . . . 12
- 2.3 WS-Resource Properties . . . 15
- 2.4 Globus Toolkit 4 . . . 19
- 2.5 Grid Computing . . . 20
- 2.6 Virtuelle Organisation . . . 23

3 Anforderungen **29**
- 3.1 Szenario . . . 29
- 3.2 Anwendung auf das VO-Modell . . . 29
- 3.3 Nichtfunktionale Anforderungen . . . 30
  - 3.3.1 Umsetzung im Rahmen des WSRF . . . 30
  - 3.3.2 Identifikatoren . . . 30
  - 3.3.3 Mitgliedschaft . . . 30
  - 3.3.4 Überlappung . . . 31
  - 3.3.5 Resource Properties . . . 31
  - 3.3.6 Erhaltung der Dynamik . . . 32
  - 3.3.7 Rahmenbedingungen der Implementierung . . . 32
- 3.4 Funktionale Anforderungen . . . 33
  - 3.4.1 Erzeugung . . . 33
  - 3.4.2 Auslesen und Manipulation der Eigenschaften . . . 33
  - 3.4.3 Auflösen der VO . . . 33
  - 3.4.4 Zugriffskontrolle . . . 33
  - 3.4.5 Fehlerbehandlung . . . 34
- 3.5 Lösungsidee . . . 34

4 State-of-the-Art **35**
- 4.1 WSRF based Virtual Organization Middleware . . . 35
- 4.2 PERMIS . . . 35
- 4.3 VOMS . . . 36
- 4.4 VOMRS . . . 37
- 4.5 Liberty . . . 37
- 4.6 Shibboleth . . . 37
- 4.7 CAS . . . 38
- 4.8 DyVOSE . . . 39
- 4.9 Zusammenfassung . . . 39

5 Entwicklung einer VO als WS-Resource **40**
- 5.1 Design der WSRF-konformen VO . . . 40
  - 5.1.1 Struktur-Design . . . 41
  - 5.1.2 Design als WS-Resource . . . 41
  - 5.1.3 Fabrik-Methode . . . 42

5.1.4 Zugriffskontrolle . . . 42
5.1.5 Identifikatoren . . . 42
5.1.6 Client . . . 43
5.1.7 Fehlerbehandlung . . . 43
5.1.8 QNames – Namespaces für Variablen . . . 43
5.2 Implementierung des Lebenszyklus . . . 44
5.3 Implementierung der VO im WSRF . . . 44
5.3.1 Factory Service . . . 44
5.3.2 Instance Service . . . 46
5.4 Implementierung der VO in GT4 . . . 49
5.4.1 VOFactoryService . . . 50
5.4.2 VOInstanceService . . . 51
5.4.3 VOResource . . . 51
5.4.4 VOMemberInfo . . . 54
5.4.5 VOMemberPermit . . . 54
5.4.6 VOClient . . . 55
5.5 Implementierung der VO . . . 56
5.5.1 VOFactoryService . . . 56
5.5.2 VOInstanceService . . . 57
5.5.3 VOResource . . . 58
5.5.4 VOResourceHome . . . 61
5.5.5 VOMemberInfo . . . 61
5.5.6 VOMemberPermit . . . 62
5.5.7 VOClient . . . 63
5.6 Build und Deployment . . . 65
5.6.1 Web Services Deployment Descriptor (WSDD) . . . 68
5.6.2 Java Naming and Directory Interface (JNDI) . . . 70
5.6.3 Namespace mappings . . . 71
5.6.4 Übersetzung und Deployment . . . 72

6 Bewertung der Lösung 73
6.1 Erreichte Ziele . . . 73
6.1.1 Management einer VO als WS-Resource . . . 73
6.1.2 Management des Lebenszyklus einer VO . . . 73
6.1.3 Management einer VO als managed object . . . 74
6.2 Mögliche weitere Entwicklungen . . . 74
6.2.1 Erweiterung zur WS-ServiceGroup . . . 74
6.2.2 Erweiterung des Factory Services um einen Zustand . . . 74
6.2.3 Ausgliedern der Personenbezogenen Informationen . . . 75
6.2.4 Erweiterung des WSRF um adressierte Manipulation . . . 75
6.2.5 Validierung der Properties . . . 75
6.2.6 Einführen eines Locking für Resouce Properties . . . 75

7 Zusammenfassung 76

I Appendix 77

A Abkürzungen 79

B WS-Standarde und -spezifikationen 81

C Erforderliche Software bei der Installation des GT4 82

D Web Services Deployment Descriptor (WSDD) 83

E Kompilieren von Web Services 84

F Deployment 86

G Starten vom Globus-Container 87

H JNDI Deployment File 89

I Beispiel für eine Endpoint Reference 90

Abbildungsverzeichnis

1.1 Das Modell einer Virtuellen Organisation [Schi 07] 2
1.2 VO Charakteristika & Fokus aktueller VO Management-Ansätze [Schi 07] 4
1.3 Zusammenhänge der Kapitel dieser Arbeit 7

2.1 Service-orientierte Architektur 9
2.2 Web Services Architektur [BHM+ 04] 11
2.3 Web Service Invocation [SoCh 06] 12
2.4 WSRF- Spezifikationen [MSB 06] 13
2.5 WSRF-ServiceGroup [MSB 06] 14
2.6 Übersicht WS-Resource 16
2.7 Komponenten vom Globus Toolkit 4 [SoCh 06] 20
2.8 Grid Architektur [Sess 03] 22
2.9 Bildung Virtueller Organisationen 24
2.10 VO Management Szenarien 26

5.1 Anwendung des Factory-Pattern zur Erzeugung von VOs [SoCh 06] 49
5.2 Nachrichtenfluss bei der Erstellung einer VO-Ressource 50
5.3 Sequenz-Diagramm zum Aufruf einer VO-Ressource [SoCh 06] 52
5.4 Resource Property Diagramm (Beispiel) 52
5.5 PDP und PIP Chain [FrAn 05] 53
5.6 PDP Auswertung [FrAn 05] 54
5.7 PDP Chain [FrAn 05] 55
5.8 GT4 Provider 58
5.9 VO Ressource Properties UML Klassendiagramm 59
5.10 Kommunikationsbeziehungen einer VO 64
5.11 Verzeichnisstruktur vom Globus Build Service (Beispiel) 66
5.12 Erzeugung eines GAR-files mit Ant [SoCh 06] 68

Tabellenverzeichnis

2.1 Lebenszyklusmanagement einer VO [SZM 06b] 26

5.1 Zusammenhang zwischen Komponenten und Klassennamen unserer Implementation 49

1 Einleitung

Grid Computing 2.5, als eine Erweiterung des bestehenden Internets, hat sich in den letzten Jahren rasant entwickelt.

Die Idee beim Grid Computing ist, die Ressourcen mehrerer Anbieter gemeinsam zu nutzen, indem sie zu einem *Grid* zusammengeschaltet werden. Die großen Datenmengen und umfangreichen Rechenaufgaben werden in kleinere Pakete verpackt und zur Auswertung weltweit auf die einzelnen Teilnehmer verteilt.

Die heutigen Anforderungen an Kapazität und Leistung können in der Zukunft fast nur durch den Aufbau von solchen Grid-Infrastrukturen erreicht werden. Um riesige Forschungs-Datenmengen auswerten zu können, reichen die Ressourcen eines einzelnen Forschungszentrums meist nicht mehr aus [RöSc 05].

Da die Erstellung eines Grids ein sehr komplexer Vorgang ist, sind diese Zusammenschlüsse auf lange Dauer angelegt. Dies steht im Widerspruch zu den unterschiedlichen und oft in kürzerer Zeit abgearbeiteten Aufgaben, die im Grid-Rahmen gelöst werden sollen.

Die wachsende Zahl und zunehmende Kapazität der weltweiten Grid-Ressourcen, die sich auch in der Auswertung einer kürzlich erstellten Umfrage unter über 20.000 weltweit verteilten IT-Fachleuten wiederspiegelt [McKe 07], erfordert eine wohldefinierte Strukturierungsmöglichkeit für den Zugriff und die Zusammenarbeit innerhalb von spontanen, zeitlich begrenzt agierenden, aufgabenorientierten Teams.

Um der Dynamik und der Kurzlebigkeit der unterschiedlichen Aufgaben Rechnung zu tragen, gibt es das Konstrukt der *Virtuellen Organisation* (VO) 2.6. Dabei werden physische Ressourcen und Personen zu einer virtuellen organisatorischen Einheit zusammengefasst. Eine solche Gruppierung ist „virtuell“, da sie unabhängig von tatsächlich vorhandenen physischen, geographischen oder sozialen Strukturen festgelegt werden kann.

1.1 Motivation und Zielsetzung

Die Virtuelle Organisationen als neue Organisationsformen entstanden im Jahre 1993 [DaMa 93]. Das grundlegende Konzept einer VO wird hier in der VO-Modell Abbildung 1.1 gezeigt. Aus der realen Welt werden Ressourcen und Dienste auf virtuelle Ressourcen und virtuelle Dienste abgebildet. Den Personen werden in der VO Rollen zugeordnet. Die Abbildungen werden dabei zu Gruppen zusammengefasst, den Virtuellen Organisationen.

Allerdings bleiben dabei viele Aspekte offen, die noch nicht geklärt sind. In der aktuellen Literatur findet man daher eine Vielzahl unterschiedlicher Definitionen für Virtuelle Organisationen, wovon sich bis jetzt keine endgültig durchgesetzt hat. Die größten Unklarheiten bei der Definition beziehen sich auf die folgende Punkte:

Mitgliedschaft von Ressourcen. Die hauptsächliche Differenz besteht in der Frage, ob Ressourcen Teil einer VO sind oder nicht. Dagegen spricht die Tatsache, dass eine Grid-Ressource nur in Verbindung mit einem Web Service 2.1 für den Nutzer einen Sinn ergibt, da dieser über den Web Service die Ressource nutzt. Daher sollte auch nur der Web Service Teil der VO sein, da er die Ressource ja implizit nutzt. Allerdings ist ein Web Service an sich ja nicht zustandsbehaftet. Eine Ressource ist das aber sehr wohl, denn sie behält ihre Konfiguration bis zur nächsten Verwendung. Das würde wiederrum für eine getrennte Behandlung von Ressourcen und Web Services sprechen, und eine Aufnahme von beiden in die VO begründen.

Die Open Grid Services Architecture (OGSA) fordert, dass Web Services einen Zustand haben können [FKS+ 06]. Das Web Services Resource Framework (WSRF) implementiert diese und bietet damit eine Lösung, die beiden Seiten gerecht wird. Wir stützen uns auf die WSRF-Definition der WS-Resource

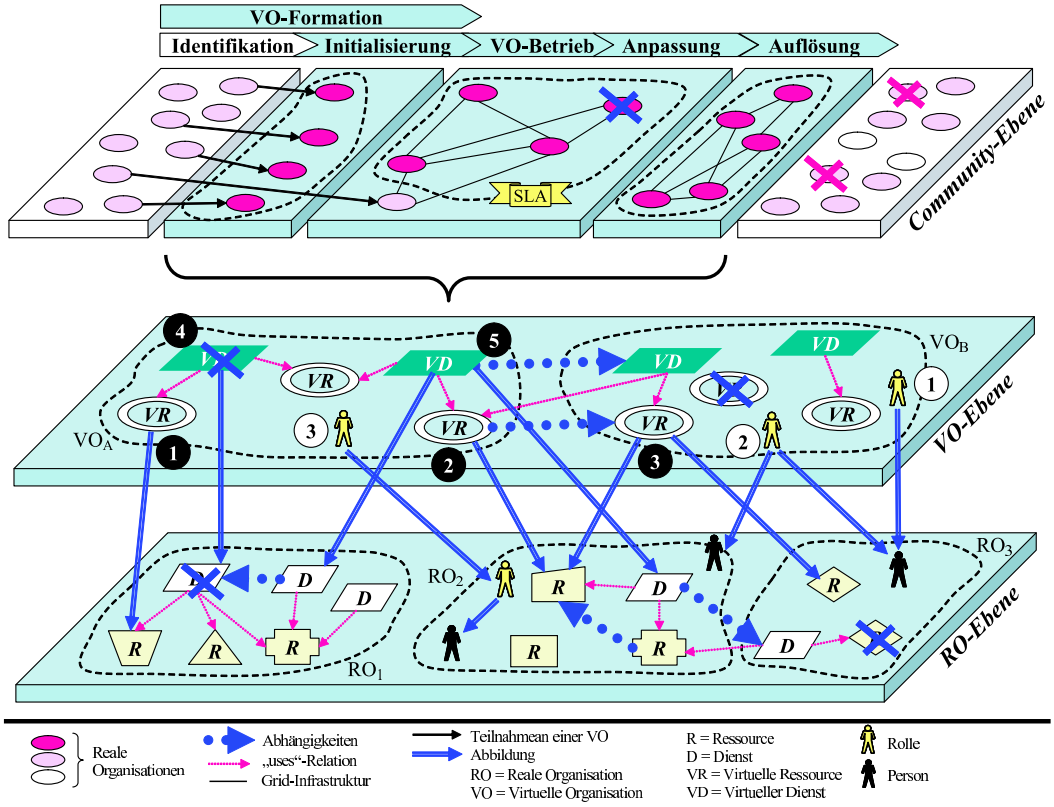


Abbildung 1.1: Das Modell einer Virtuellen Organisation [Schi 07]

[GKM+ 06]. Diese enthält neben der Angabe eines Web Services auch optional die Angabe der verwendeten Ressource. Somit können wir sowohl tatsächlich zustandslose Web Services aufnehmen, als auch zustandsbehaftete, die nur in Verbindung mit einer Ressource Bestand haben können.

Mitgliedschaft von Personen. Dass Personen in der ein oder anderen Form zu einer VO gehören müssen, ist unstrittig. Das VO-Modell 1.1 sieht vor, dass einer VO nur Rollen angehören. Die Personen aus der realen Welt werden dann auf Rollen in der virtuellen Welt abgebildet.

Diese Abbildung auf Rollen senkt den Administrationsaufwand, fügt aber eine zusätzliche Abstraktionsschicht ein. Der Aufwand ist bei der Aufnahme oder dem Entfernen einer Person wesentlich geringer, da nur die Rollenzuordnung gelöscht werden muss. Die Berechtigungen und Verknüpfungen mit Ressourcen und Diensten müssen nicht extra gepflegt werden. Dafür muss aber die Abbildung der Personen über die Rollen auf die Komponenten der VO stets im Auge behalten werden, damit keine unbeabsichtigten Verknüpfungen entstehen.

Management des Lebenszyklus einer VO. Ein hoher Grad an Dynamik ist eine Basiseigenschaft von Virtuellen Organisationen. Diese richtet sich sowohl nach „Innen", als auch nach „Außen". Das bedeutet, dass eine VO jederzeit erzeugt oder aufgelöst werden kann. Auch können neue Teilnehmer aufgenommen oder bestehende entlassen werden.

Die Tatsache, dass diese Organisationen jederzeit entstehen oder aufgelöst werden können, bezeichnet man auch als den Lebenszyklus (siehe auch Abschnitt 2.6. Dabei werden bei jeder Transition im Lebenszyklus einer VO die Abbildungen der realen in die virtuelle Welt, wie in Abbildung 1.1 gezeigt, verändert. Die Frage nach den Gesetzmäßigkeiten dieser Veränderungen entspricht der Frage nach dem speziellen Management des Lebenszyklus einer VO. Dieser Aspekt wird in [Schi 07] genauer betrachtet.

Die theoretischen Aspekte der Verwaltung des Lebenszyklus sind dabei geklärt worden. Allerdings gibt es noch keine konkrete Umsetzung oder Standards zur Implementierung von entsprechenden Werkzeugen.

Sicherheitsmanagement. Unter dem Begriff *Sicherheitsmanagement* werden alle Maßnahmen/ Regelungen zusammengefasst, die in einer Organisation vorgesehen sind, um ein hohes Sicherheitsniveau zu erreichen. In einer VO wird eine umfassende Gruppe von Personen, Web Services und Ressourcen zusammengeschlossen. Gemäß dem VO-Modell, dargestellt in 1.1, werden Personen in eine VO auf Rollen abgebildet. Durch das Rollenkonzept ergeben sich unterschiedliche Berechtigungsklassen für den Zugriff auf virtuelle Ressourcen und Dienste.

Diese Implementierung der Sicherheitsaspekte sind daher gesondert zu betrachten und umzusetzen. Einige aktuelle Ansätze betrachten VOs ausschließlich bezüglich ihrer Berechtigung. Diese Ansätze werden in Kapitel 4 ausführlicher erklärt. Daraus geht auch hervor, dass die meissten Ansätze sich rein auf das Sicherheitsmanagement konzentrieren. Die Gründung und Verwaltung von VOs wird überwiegend auf das Verwalten von Rechten und Rollenzuordnungen beschränkt.

In unserer Umsetzung werden wir uns vornehmlich auf das Lebenszyklusmanagement konzentrieren, also die Erzeugung, Bearbeitung und Auflösung von VOs (siehe Grafik 1.2). Das Sicherheits- [HAN 99] und das Fehlermanagement [HAN 99] werden unter sekundärem Fokus betrachtet. Auf Accounting [HAN 99] und Performance [HAN 99] wird in dieser Arbeit nicht eingegangen.

Wir definieren für die Arbeit wie folgt:

- Virtuelle Organisationen fassen eine Untermenge der im Grid enthaltenen Ressourcen, Dienste und Personen für eine begrenzte Zeit unabhängig von realen Strukturen zusammen.
- Personen sind Nutzer des Grids, die sich mit einem Client über Maschine–Maschine–Kommunikation gegenüber dem Grid authentifizieren und auf dessen Web Services zugreifen. Sie werden in eine VO über Rollen abgebildet.
- Jeder potentielle Grid-Teilnehmer wird durch Mechanismen des Grids authentifiziert, bevor er Operationen auf Ressourcen innerhalb des Grids ausführen kann.
- Auch eine VO kann in eine andere VO aufgenommen werden, sie wird zur Sub-VO. Durch diese Mitgliedschaft können den Mitgliedern der Sub-VO die Dienste und Ressourcen der übergeordneten VO zugänglich gemacht werden.
- Sub-VOs sind VOs, die einer zweiten VO untergeordnet sind. Dadurch erben sie die WS-Resourcen der Über-VO, und können auf sie zugreifen.

Zwei Non-Profit Organisationen kümmern sich um die Standardisierung der Grid relevanten Themen: das Global Grid Forum (GGF) [1] und die Organization for the Advancement of Structured Information Standards (OASIS) [2]. Das GGF hat sich dabei eher mit abstrakten Themen wie Architektur [FKS+ 06] und Infrastruktur [TCF+ 03] beschäftigt, die OASIS mit Details wie Transport, Zugriff und Beschreibungen von Diensten. Um beide Entwicklungen zusammenzuführen, haben sie schließlich gemeinsam am Web Services Resource Framework (WSRF) 2.2 gearbeitet, in dem die Entwicklungslinien der Grid Standards und der webbasierten Standards zusammengeführt wurden. Auf dieser Grundlage werden wir unsere Arbeit aufbauen.

Die Entscheidung für die Umsetzung von VOs als WSRF-konforme Ressourcen ist naheliegend. Allerdings gibt es bis heute noch keine ähnliche Implementierung (siehe auch [Schi 07]). Die bereits existierenden Mittel zur Verwaltung von bisherigen Ressourcen können ohne Anpassung weiterverwendet werden. Es ist daher nicht eine neue Management-Infrastruktur mit zusätzlichen Werkzeugen einzurichten, sondern bestehende Elemente und Prozesse können direkt übernommen werden.

Die Umsetzung wird unter Verwendung des Globus Toolkits 4 (GT4) 2.4 erfolgen, das im Grid-Bereich als de-Facto Standard gilt [Manh 06]. Seit Version 4 basiert es vornehmlich auf dem WSRF [uDTB 07], welches wie oben erwähnt Implementationsgrundlage dieser Arbeit ist.

[1] Weitere Informationen unter http://www.ggf.org/
[2] Weitere Informationen unter http://www.oasis-open.org

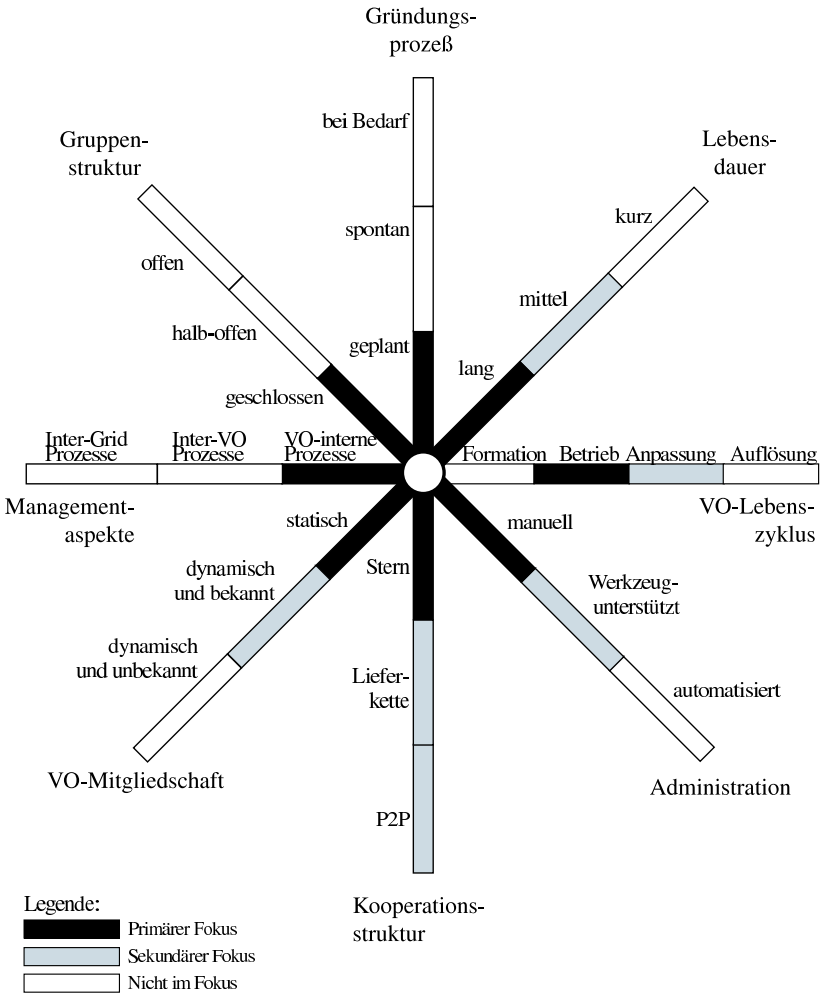


Abbildung 1.2: VO Charakteristika & Fokus aktueller VO Management-Ansätze [Schi 07]

1.2 Fragestellung

Wie oben schon erwähnt, bietet die Virtuelle Organisation eine essentielle Strukturierungsmöglikeit im Einsatz von Grid-Computing. Die Gruppierung von Personen und Ressourcen zu einer logischen, dynamischen und flexiblen Einheit bietet die einzigartige Möglichkeit, die Zusammenarbeit unter der Nutzung von verteilten Ressourcen effizient zu gestalten.

Ressourcen und Dienste haben in der Realität eine feste geographische Position, und werden von der Organisation verwaltet, in dereen Eigentum sie sind. Auch bei dem Zusammenschluss zu einem Grid bleibt diese Lokalität erhalten. Werden sie nun, gemäß dem VO-Modell aus der Abbildung 1.1, in die Virtualität abgebildet, verschwimmen die geographischen und organisatorischen Grenzen. Daher bekommt nun die Verwaltung virtueller Ressourcen und Dienste einen neuen, globalen Charakter. Dabei muss stets die Brücke zwischen dem Management der realen Ressourcen und deren virtuellen Abbild geschlagen werden.

Virtuelle Organisationen fassen eine Untermenge der im Grid enthaltenen Ressourcen und Personen für eine begrenzte Zeit in eine neue Organisationsform zusammen. Während dieser „Lebenszeit“ kann sich ihre Konfiguration [3] jederzeit ändern. Diese Dynamik stellt hohe Anforderungen an die Verwaltung von VOs (siehe auch Kapitel 3, da hierbei nun tatsächlich organisatorische, geographische und politische Grenzen durchgebrochen werden.

[3] Konfiguration = Summe aller Eigenschaften eines Systems

Das Management virtueller Organisationen ist daher Gegenstand jüngster Forschung. Hauptsächliche Fragen dabei sind:

1. Welche Managementanforderungen werden im Rahmen einer VO gestellt?
2. Wie kann man Virtuelle Organisationen implementieren?

Die erste Frage wurde dabei ausgiebig in [Schi 07] betrachtet. Darauf aufbauend wollen wir in dieser Arbeit einen der möglichen Ansätze zur Lösung der zweiten Frage untersuchen. In dieser Arbeit wird das Ziel verfolgt, bestehende Standards zum Ressourcen-Management auch für die Verwaltung Virtueller Organisationen verwenden zu können. Das hierbei verwendete Web Services Resource Framework (WSRF) wird im Abschnitt 2.2 genauer beschrieben. Alternative Lösungswege werden in Kapitel 4 vorgestellt.

Die konkrete Fragestellung dieser Arbeit lautet wie folgt:

1. Inwieweit kann das WSRF-Rahmenwerk zur Realisierung von VOs in Grids verwendet werden?
2. Welche Lücken müssen geschlossen werden bzw. mit welchen Einschränkungen muß man zurecht kommen?

Die Beantwortung dieser Fragen ist von nicht unerheblicher Bedeutung für das Management Virtueller Organisation in Grids (siehe [Schi 07]), da WSRF die Grundlage aller Web Services Management-Ansätze bildet (siehe auch [BMW 06] und [CKM+ 03]).

Web Services Management-Ansätze sind jedoch nicht Gegenstand dieser Arbeit. Der interessierte Leser sei auf die angegebenen Spezifikationen verwiesen [Schi 07].

1.3 Vorgehensmodell

Im Folgenden wird die Struktur dieser Arbeit dargestellt. Der Zusammenhang der Kapitel ist in der Abbildung 1.3 zum besseren Verständnis visualisiert.

Die grundlegende Begriffe werden im Kapitel 2 eingeführt. Zunächst wird in Abschnitt 2.1, soweit es für die Belange dieser Arbeit relevant ist, eine generelle Architektur von Web Services, sowie deren Vor- und Nachteile gegenüber anderen Technologien dargestellt. Außerdem wird dort erklärt, wie ein Web Service-Aufruf stattfindet.

Da Ziel dieser Arbeit die Realisierung einer VO mittels Web Services Resource Framework (WSRF) ist, wird im nächsten Abschnitt 2.2 das auf Web Services basierende WSRF ausführlich erklärt. Das Framework bietet die Möglichkeit, Web Services mit einem Zustand zu versehen, der über ein standardisiertes Interface manipuliert und abgefragt werden kann. Die Zustände werden in Ressourcen gespeichert und jeweils in Kombination mit einem Web Service verwendet. Diese Spezifikation ist einer der Kernbausteine dieser Arbeit.

Jede WS-Resource wird durch ihre Eigenschaften beschrieben. Da wir in unserer Arbeit eine Virtuelle Organisation als WS-Resource implementieren wollen, ist es wichtig die Properties von WS-Resourcen zu betrachten. Im Abschnitt 2.3 werden die Arten von Eigenschaften einer WS-Resource, sowie die Nachrichten-Formate der Anfragen und Antworten bei Operationen auf diese dargestellt.

Teil der Aufgabenstellung dieser Arbeit ist, die Umsetzung im Kontext des Globus Toolkits 4 zu untersuchen. Deswegen folgt im Abschnitt 2.4 eine kurze Beschreibung dieses Toolkits. GT4 bietet eine Technologie zur Erstellung von Grids und ist eine Vermittlung zwischen Anwendungs- und System-Schicht. Um ein besseres Verständnis zu gewährleisten, wird die Globus-Architektur schichtenweise betrachtet.

Da wir Virtuelle Organisationen in Grids aufbauen wollen, wird im Kapitel 2.5 das Grid Computing Konzept dargestellt. Dazu gehören die Darstellung der Grid-Architektur in Schichten, sowie eine kurze Beschreibung von Grid-Systemen, worauf unsere VO-Strukturen gebildet werden.

Kerngedanke dieser Arbeit sind die Virtuellen Organisationen. Diese dynamische Grid-Einheiten werden im Abschnitt 2.6 ausführlich beschrieben. Dabei wird auf Themen wie Sicherheit und Management einer VO eingegangen. Welche Merkmale eine VO besitzt und wie eine VO realisiert werden kann, werden ebenfalls in diesem Abschnitt diskutiert.

Nach diesen Grundlagen werden im Kapitel 3 aus einem realen Szenario die Anforderungen an die angestrebten Lösung abgebildet.

Eingangs wurde bereits erwähnt, dass auch anderenorts die Implementation Virtueller Organisationen Forschungsgegenstand ist. Im Kapitel 4 werden die wichtigsten alternativen Ansätze zusammen mit einer Beschreibung der Unterschiede und Gemeinsamkeiten aufgezählt.

In dem vorletzten Kapitel 5 ist die Umsetzung der tatsächlichen Web Services zur Erstellung und zum Managen von Virtuellen Organisationen zu finden. Mehrere Komponenten sind bei der Kommunikation zwischen dem Client und dem WS-Anbieter involviert. Jeder einzelnen Komponente wird je ein Unterkapitel gewidmet. Um die entwickelten Funktionen zu testen, wurde zusätzlich ein Client Programm erstellt, mit dem die Verwaltung unserern VOs und somit der „Proof of Concept“ gelingt.

Im Kapitel 6 wird die Implementierungslösung bewertet, es werden dabei die Vorteile der Implementierung beschrieben, sowie die Probleme, die während der Arbeit entstanden sind, dargestellt. Weitere Entwicklungsmöglichkeiten sind auch vorgeschlagen. Eine Zusammenfassung der Arbeit erfolgt abschließend im Kapitel 7.

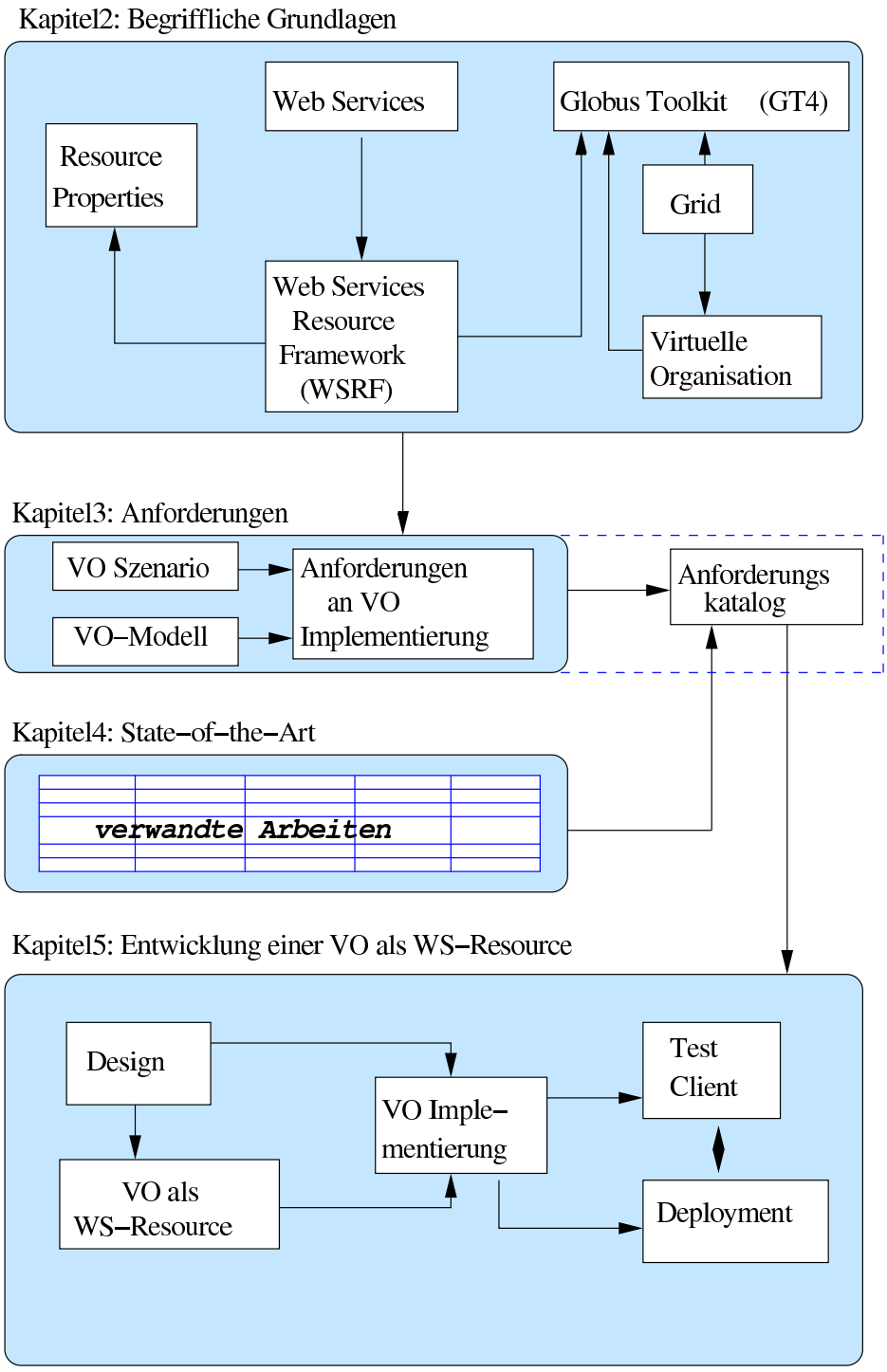


Abbildung 1.3: Zusammenhänge der Kapitel dieser Arbeit

2 Begriffliche Grundlagen

Bevor wir die Fragen aus der Einleitung klären können, wollen wir den Leser mit verschiedenen thematisch relevanten Begriffen vertraut machen. Diese entspringen größtenteils der anerkannten wissenschaftlichen Literatur. An einigen Stellen gibt es aber, wie in der Einleitung bereits beschrieben, noch große Kontroversen. In diesem Kapitel werden wir also nur die für uns relevanten Definitionen beschreiben.

Wie wir einleitend gesehen haben, sind Web Services 2.1 ein Grundelement von Grids. Sie sind das Portal für den Anwender, der mit Hilfe seiner Client-Software über definierte Schnittstellen und Protokolle seine Aufträge und Anfragen an Web Services absetzt. Daher werden wir zunächst die Bestandteile eines Web Services, sowie die Spezifikationen, die ihn definieren, erklären.

Darauf aufbauend betrachten wir das Web Services Resource Framework (WSRF) 2.2, in dessen Kontext wir die VO implementieren. Dabei sehen wir uns die verschiedenen Standards an, aus denen sich das WSRF zusammensetzt. Einer davon spielt für die vorliegende Arbeit eine zentrale Rolle, da er die Eigenschaften einer Ressource beschreibt. Er wird daher in einem separaten Abschnitt 2.3 ausführlich erklärt.

Das Globus Toolkit 4 2.4 implementiert das Web Services Resource Framework. Da unsere Umsetzung Virtueller Organisationen auf GT4 basieren wird, werden wir diese Software im darauf folgenden Abschnitt darstellen.

Nachdem die Basistechnologien damit geklärt sind, erheben wir uns ein wenig von der Technik und betrachten die Zusammenhänge und Anforderungen des Grid Computings 2.5 im Allgemeinen.

In der Einleitung haben wir bereits gesehen, dass die Bildung Virtueller Organisationen 2.6 ein wichtiger Bestandteil des Grid Computings ist. Wir werden daher zum Abschluss des Grundlagenkapitels die Aspekte dieses besondere Kostrukts untersuchen.

2.1 Web Services

Um eine einfache Kommunikation zwischen unterschiedlichen Software-Systemen und Applikationen im Internet zu gewährleisten, werden so genannte *Web Services* entwickelt, eine Technologie aus dem Bereich *distributed computing* [Rose 98], die unabhängig vom Betriebssystem oder der Programmiersprache ist [Hais 03].

Viele Definitionen von Web Services stellen das Grund-Charakteristikum von webbasierten Services dar. Das W3C-Konsortium [1] beschreibt diese Dienste wie folgt:

> *„Unter einem Web Service (auch: webbasierten Service) versteht man eine eigenständige, selbst beschreibende und modulare Software Applikation, die im Internet veröffentlicht, lokalisiert und aufgerufen werden kann. Die Funktionalität eines Web Services kann von einfachen Anwendungen bis hin zu komplexen Business Prozessen reichen.“* [WK 06]

Eine andere WS-Definition vom W3C-Konsortium, aus dem Web Services Description Language (WSDL)-Primer [WK 06] entnommen, beschreibt die Web Services aus einer anderen Sicht, und zwar, aus der Sicht der Anforderungen, die bei dem Einsetzen von Web Services zu erfüllen sind:

> *„A Web Service is a software system designed to support interoperable machine-to-machine interaction over network.“* [WK 06]

[1] Weitere Informationen unter http://www.w3.org/

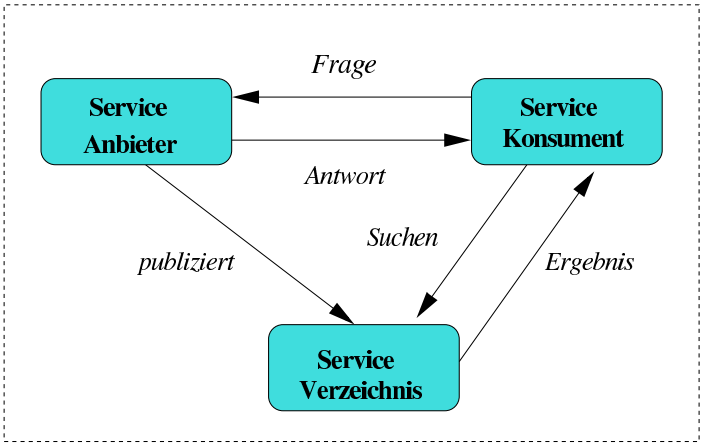


Abbildung 2.1: Service-orientierte Architektur

Ein Web Service beschriebt also eine Schnittstelle, die verschiedene Methoden und Operationen bietet, die durch XML-Dokumente aufgerufen werden können. Die einzige Voraussetzung für die Kommunikation zwischen einer in einer beliebigen Programmiersprache geschriebenen Software und den Web Services ist, dass die gesendete Anfragen in XML [Jeck 04] beschrieben sein müssen. Zu betonen ist, dass die Programme eines Web Services im Vergleich zu den anderen also über ein XML-gekapseltes Protokoll kommunizieren.

Die Kommunikation an sich verläuft auf einem anderen Niveau als im Internet, denn es werden keine Hypertext Transfer Protocol (HTTP)-Seiten an den Webbrowser geschickt, die dann von den Menschen betrachtet werden. Programme tauschen hier Daten aus und starten benötigte Funktionen auf entfernten Rechnern [Thor 05].

In einem Grid können mehrere Clients auf einen und denselben Web Service zugreifen. Deswegen muss sicher gestellt werden, dass jeder Client zu jeder Zeit auf die gewünschte Services zugreifen kann, bzw. dass genügend Ressourcen für jeden Client zur Verfügung stehen. Alle Clients können im „worst case“ auf denselben Web Service zugreifen. Dieser und weitere „use cases“ müssen bei der Verwaltung von Web Services bzw. Ressourcen berücksichtigt werden. Ein solcher Web Service, auf den mehrere Nutzer parallel zugreifen können, oder woran ein Nutzer mehrere unabhängige Anfragen gleichzeitig abschicken kann, heißt *Multiple Resource Web Service [Akra 06]*.

Die „zustandslose“ Services basieren auf verbreitete Technologien, wie IP-Protokolle für den Zugriff auf diese, oder XML für die Verschlüsselung von abgehenden Requests/ entgegenkommenden Responses. Somit können verschiedene Applikationen unabhängig von der Plattform miteinander interagieren.

Die Interoperabilität basiert auf einem Modell, das Ähnlichkeiten mit dem Client-Server-Modell hat: jede Applikation besteht aus einem „Lieferanten“ (Server) und einem oder mehreren „Verbraucher(n)“ (Clients). Damit der Client den angebotenen Server-Dienst „versteht“, bietet der Web Service unterschiedliche Funktionen, die in so genannten Web Services Description Language (WSDL) [WK 06]-Dokumenten definiert sind. Die Kommunikation zwischen Web Services, oder zwischen dem Web Service und dem Client verläuft meistens mit Hilfe des Simple Object Access Protocols (SOAP) [MiLa 07], dem gegenüber anderer Kommunikationsarten diverse Vorzüge eingeräumt werden.

Die für die Web Services verwendete Service-orientierte Architektur (SOA) [He 03] basiert auf drei Hauptkomponenten: Serviceanbieter, -verzeichnis und -konsumenten. Der Anbieter bietet seine Web Services, die er in einem bestimmten Verzeichnis (eine Art „Gelbe Seiten“) einträgt. Diese werden weiterhin vom Konsumenten nach Bedarf ausgewählt. In der Abbildung 2.1 sind die Beziehungen zwischen diesen Teilnehmern verdeutlicht.

Die weitere Vorteile, sowie die Nachteile vom Web Services Einsatz werden wir im Folgenden kurz beschreiben [MiLa 07].

Die *Vorteile* von Web Services gegenüber anderer Technologien wie Common Object Request Broker Architecture (CORBA) [Wall 07], Remote Method Invocation (RMI) [Sun 03], Enterprise JavaBeans (EJB)

[Sun 06], Distributed Computing Environment (DBE) [Rose 98] etc. beeinflussen enorm bei der Entscheidung zur Auswahl der Middleware für die Applikationssysteme. Heutzutage werden die Web Services als eine Weiterentwicklung von den oben benannten Technologien gesehen:

- Web Services sind *plattform- und sprachenunabhängig*. Jeder Client kann damit in verschiedenen Sprachen, anders z.B. als der Service, geschrieben werden. Bei den anderen Technologien haben sich die Entwickler für eine bestimmte Programmiersprache geeignet und damit Grenzen gesetzt. Die Web Services sind unabhängig und universell einsetzbar.
- Die Nachrichten-Kommunikation (Request/ Response) verläuft meistens mit Hilfe vom *HTTP-Protokoll*. Deswegen tretten bei der Übertragung von Daten sehr selten Probleme mit der Firewall, wie bei CORBA oder RMI. Manchmal werden außer HTTP andere Protokolle, wie File Transfer Protocol (FTP) oder Simple Mail Transfer Protocol (SMTP), verwendet. Ein Web Service kann also im Internet, wie Intranet oder Extranet [2]-Umgebung genutzt werden.
- Ein Web Service basiert auf *SOA-Architektur*, d.h. jeder Dienst wird in Teildiensten aufgeteilt und aufgeführt, wobei jeder Teilprozess unabhängig von den anderen ist. Damit stellt sich sicher, dass die Web Services für verschiedene Zwecke wiederverwendbar und wartbar (wegen Unabhängigkeit von Services) sind.
- Die Web Services bieten eine klar definierte Zugriffsschnittstelle.
- Die wichtigsten WS-Standards (siehe Anhang B) und Protokolle basieren auf XML, somit werden die Vorteile einer XML-Beschreibung ausgenutzt, beispielsweise stehen die offene Standards jedem zur Verfügung.

Neben den oben genannten Vorteilen sind beim Einsetzen von Web Services im Zusammenhang mit Grid Computing auch einige Nachteile wie folgt zu erwähnen:

- Die Web Services selber sind *zustandslos*, d.h. Zustände können nicht gespeichert werden. Der Client kann deswegen auf eine und dieselbe erzeugte Instanz bei dem erneuten Methoden-Aufruf nicht mehr zugreifen. Wenn ein Client als Beispiel mehrmals den Web Service aufruft, muss er mit mehreren Instanzen und nicht mit einer einzigen arbeiten.
- Durch das Einsetzen von XML wird natürlich die Portabilität erreicht. Die Effizienz und Performance sinken dabei aber deutlich (z.B. wegen Overhead) [Jeck 04].
- Die Sicherheit beim Datentransport in der webbasierten Kommunikation ist einer der wichtigsten Aspekte überhaupt. XML ist grundsätzlich textbasiert und nicht verschlüsselt. Ohne zusätzliche kryptographische Hilfsmittel können Dritte den Nachrichtenstrom problemlos mitverfolgen.
- Damit bei der Web Service Technologie verschiedene Programmiersprachen eingesetzt werden können, sind sog. *Biblioteken* erforderlich.

Wie in der Abbildung 2.2 gezeigt wird, ist die WS-Architektur in vier Service-Schichten aufgeteilt [BHM+ 04]:

Service Processes. Dieses Layer bezieht sich auf verschiedene Prozesse, die von mehreren Web Services unterstützt werden. Als Beispiel nehmen wir den Discovery Service, ein Verzeichnisdienst, der die für den Client angebotenen Web Services verwaltet.

Der Client fragt den Discovery Service nach einem bestimmten Web Service. Als Anwort bekommt er die eigentliche WS-Adresse (als Uniform Resource Identifier (URI)), über die er dann die Kommunikation zu dem entsprechenden Dienst aufnimmt. Meinstens ist für den Verzeichnisdienst ein extra-Server eingerichtet.

Noch zu beachten ist, dass kein direkter Kontakt zwischen dem Benutzer und dem Web Service möglich ist. Deswegen sind URIs auch nicht für den Browser gedacht, sondern sie werden in den Programmen eingebaut, die auf die Services zugreifen wollen.

Service Description. Eins der wichtigsten Merkmale von Web Services besteht darin, dass die Web Services sich *selbst* beschreiben können. Durch eine Beschreibung in der Web Services Description Lan-

[2] Teil vom Intranet, dass von einer geschlossenen Benutzergruppe vom Außen erreichbar ist

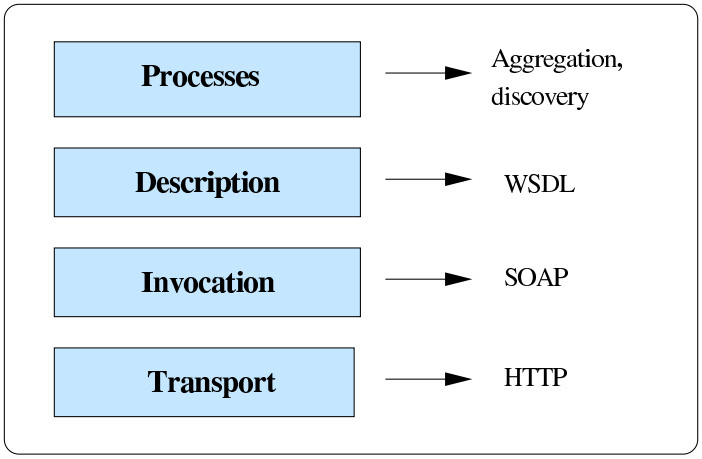


Abbildung 2.2: Web Services Architektur [BHM+ 04]

guage (WSDL) werden die auf dem Server abgelegte Services ihre unterstützende Funktionen eindeutig spezifiziert.

Service Invocation. Diese Schicht bezieht sich auf die Client-Server-Kommunikation und gibt das Format vor, wie die Anfragen von den Clients und die Antwort-Nachrichten vom Server auszusehen haben. Das Simple Object Access Protocol (SOAP) [Thor 05] ist das meinst benutzte Kommunikationsprotokoll bei den Web Services, da es die Möglichkeit gibt, typisierte und strukturierte Informationen über das Internet auszutauschen.

Service Transport. Das verbreitetste Transport-Protokoll bei den Web Services ist das HTTP, dasselbe, das auch für den Zugriff von Web-Seiten im Internet zuständig ist [SoCh 06].

Im Folgenden werden wir einen einfachen Aufruf von Web Services beschreiben, sowie die Rolle von WSDL, SOAP und Stubs herleiten.

Bei einem WS-Request bei der Discovery Komponente wird der Dienst lokalisiert und anschließend die URI-Adresse an den Client zurückgeschickt. Auf Basis von WSDL-Beschreibung von dem Web Service werden entsprechende Stubs erzeugt. Die Stubs müssen nicht immer neu erzeugt werden, sondern nur einmal, beim Kontaktieren der Discovery-Komponente. Dank der Stubs müssen die Clients nicht daran denken, wie man SOAP-Anfragen zu generieren und SOAP-Antworten (vom Server) zu interpretieren sind. Genauso für den Server bleibt das Interpretieren von den SOAP-Requests oder die für den Client generierende SOAP-Responses transparent.

Wie ein WS-Aufruf stattfindet, zeigt uns die Abbildung 2.3. Im Folgenden werden wir eine übliche WS-Invocation (Deutsch: WS-Aufruf) schrittweise erklären [SoCh 06].

1. Der Client ruft die gewünschte Methode auf den bei ihm lokal vorliegenden Stubs auf. Diese kapseln den Aufruf in eine SOAP Nachricht. Dieser Prozeß heißt *Serialisierung*.

2. Die SOAP-Nachricht wird übers Netz (HTTP-Protokoll) an die Server-Seite geschickt. Die Server Stubs übersetzen die Nachricht erneut in einen Methodenaufruf. Dieser Umkehrprozeß heißt *Deserialisierung*.

3. Nach der *Deserialisierung* wird Web Service Invocation schließlich ausgeführt. Die gewünschte Methode des angesprochenen Objektes wird aufgerufen und das Ergebnis entgegengenommen.

4. Die Ergebnisse werden von den Server Stubs in der SOAP-Antwort eingebettet.

5. Im Folgenden wird die Nachricht über das Internet zurück an den Client weitergereicht. Die Stubs übersetzen die Ergebnisse, so dass die Client-Applikation sie interpretieren kann.

6. Die Applikation nimmt die Ergebnisse entgegen und verwendet sie weiterhin für die eigene Zwecke.

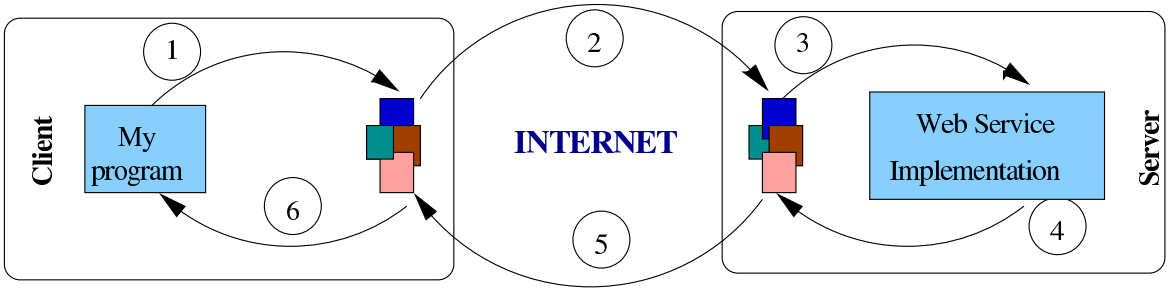


Abbildung 2.3: Web Service Invocation [SoCh 06]

Wie in der Einleitung schon erwähnt, werden die benötigten Web Services in unserer Arbeit auch implementiert. Damit besteht die Möglichkeit VOs anzulegen, zu betreiben und zu managen. Dabei setzen wir den auf WS aufbauenden Standard, das Web Services Resource Framework (WSRF) ein, der in dem nächsten Abschnitt erklärt wird.

2.2 Web Services Resource Framework (WSRF)

Das Ziel eines Standards beruht auf die Unterstützung beim Zusammenschluß von dezentralen und heterogenen Systemen zu gewährleisten, wobei die Plattformen, Sprachen und Implementierungen unterschiedlich sind.

Durch die entwickelten Standards wird versucht die Kompatibilität, gemeinsame Nutzung, Portabilität und Erweiterung von Grid-Systemen zu erleichtern.

Organisationen und Konsortien wie das Global Grid Forum (GGF), die Globus Alliance[3] und die Organization for the Advancement of Structured Information Standards (OASIS) beschäftigen sich mit der Entwicklung von Grid-Standards, wie Open Grid Services Architecture (OGSA) [FKS+ 06], Open Grid Services Infrastructure (OGSI) [TCF+ 03], Web Services Resource Framework (WSRF) [Tim 05], Universal Description, Discovery und Integration (UDDI) [CHvRR 04] etc.

Das WSRF ist eine Sammlung modularer Einzelspezifikationen und beschreibt, wie mit Hilfe von Web Services zustandsbehaftete Ressourcen abgefragt, geändert und repräsentiert werden können. Die Spezifikationen von WSRF beinhalten also verschiedene Pattern für die Definition, Operationen, Fehlerbehandlungen und Lebenszyklen von Ressourcen [GKM+ 06].

Das WSRF wird als Weiterentwicklung von OGSI gesehen; dabei wird leicht die Syntax und die Terminologie geändert, um eine bessere, logischere Auftelung von Spezifikationen zu schaffen. Außerdem wurden die eigenen Standards um die *WS-Notification*- und *Addressing*-Spezifikation erweitert [GrTr 06].

Das Grundziel des WSRF war es, eine Kommunikation zwischen verschiedenen Web Services, die auf Ressourcen zugreifen, zu definieren, sowie eine Wiederverwendbarkeit von Ressourcen zu sichern.

Neben vielen anderen Standards bildet also das Framework eine Basis, auf der die angebotene Web-/ Grid-Services aufbauen. Da die Web Services von Natur aus zustandslos sind, wurde das Simulieren von zustandsbehafteten Diensten in Acht genommen. Die Lösung dafür ist nun das Verwenden eines Tokens, das bei jeder Anfrage (Request) mit der SOAP-Nachricht mitgesendet wird. In diesem Framework bleibt das „Verwenden des Tokens“ für jeden Client versteckt bzw. transparent. Der Token ist auch als *Endpoint Reference (EPR)* bekannt. Bei der Erzeugung einer Ressource wird also der EPR an den Client als Antwort geschickt, das bei jedem Methoden-Aufruf in der SOAP-Nachricht eingebettet wird. Die Transparenz wird mit Hilfe vom WS-Addressing (WSRF-Teilstandard) geschafft [GHR 06].

In diesem Abschnitt werden wir jede einzelnen Spezifikation ausführlicher eingehen. Dabei werden wir die wichtigsten Merkmale und vorgeschlagene WSRF-Methoden in Betracht ziehen.

[3] Weitere Informationen unter http://www.globus.org

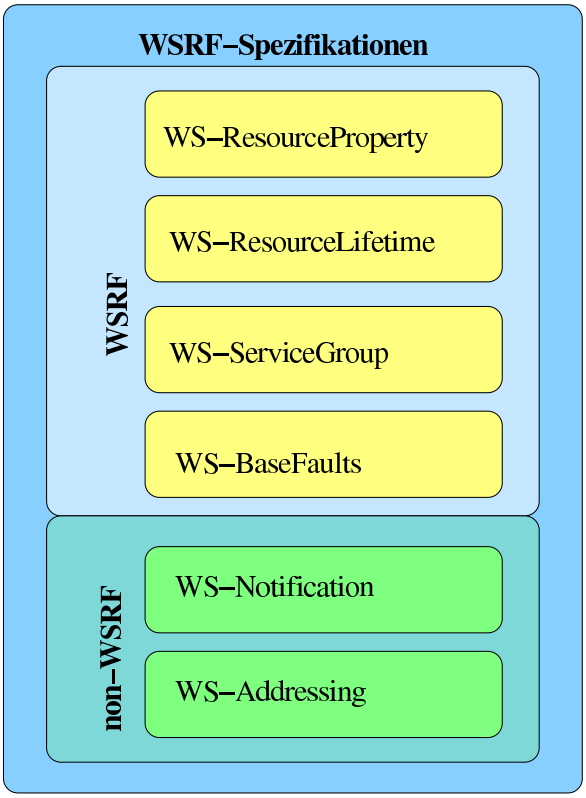


Abbildung 2.4: WSRF- Spezifikationen [MSB 06]

Im Vergleich zu OGSA bietet alsals das Framework eine bessere Übersichtlichkeit und eine logische Aufteilung nach Themenbereichen [ReSc 04]. Die WSRF-Entwickler haben sich auf folgende *WSRF* und *non-WSRF* Einzelspezifikationen, die sich mit der Zeit zu eigenen Standarden gebildet haben, geeinigt [Kner 06]:

WS-ResourceLifetime. Um die Lebenszeit der WS-Resource zu setzen, abzurufen, oder den Lebenszyklus von erzeugten Instanzen zu kontrollieren, werden Operationen dieser WSRF-Teilspezifikation verwendet. Es wird zwischen *zeitgesteuerter* und *sofortiger* Zerstörung unterschieden. Der Zeitpunkt der Zerstörung darf beliebig weit in der Zukunft liegen.

Nachdem ein Client eine Ressource aufgefordert hat, wird diese nach einer bestimmten Zeit für die anderen Clients freigegeben. Die *sofortige* Zerstörung wird also vom Client ausgelöst, die *zeitgesteuerte* durch den Server. Bei Bedarf muss vom Client die Lebensdauer einfach verlängert werden. Um die Lebensdauer zu verlängern oder zu verkürzen, wird die WSRF-Methode `setTerminationTime` verwendet. Interessiert uns die Referenzzeit des Servers, wird die Funktion `CurrentTime` aufgerufen. Eine sofortige Zerstörung einer WS-Resource wird mit der `destroy`-Methode erreicht. Nach der Ausführung jeder Aktion, ob erfolgreich oder fehlerhaft, wird immer eine Nachricht an den Client zurückgeschickt [SrBa 06].

WS-ResourceProperties. Diese Teilspezifikation beschreibt Operationen, die den Zugriff auf die ResourceProperties erlauben. Sie bietet die Möglichkeit einzelne oder mehrere Eigenschaften einer Ressource zu ändern oder auch Zustandswerte von Ressourcen abzufragen.

Mit `Get-, Set-` oder `DeleteResourceProperty`-Methoden können einzelne Properties zurückgegeben, gesetzt oder gelöscht werden. Außerdem müssen alle Properties einer Ressource in einem Resource Property Document (RPD) (siehe auch 2.3) gespeichert werden.

Um alle Eigenschaften des *Resource Property Document*s gleichzeitig zu modifizieren, werden WSRF-Methoden verwendet, wie `Get-` und `PutResourcePropertyDocument`. Diese Methoden wurden eingeführt, damit die Anzahl der verschickten Nachrichten gering bleibt. Somit können bei einer An-

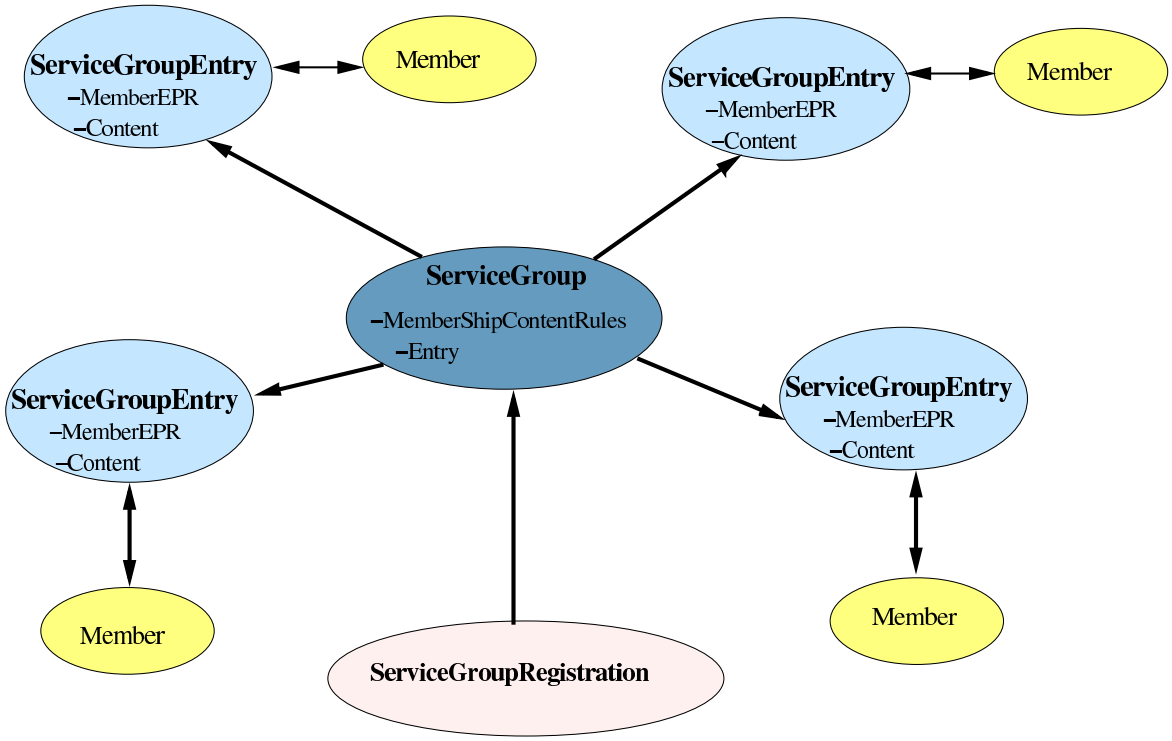


Abbildung 2.5: WSRF-ServiceGroup [MSB 06]

frage mehrere Werte gleich ausgelesen und ausgegeben werden. Die Zustandswerte können auch mit Hilfe von einem XPath-Ausdruck als Parameter von `QueryResourceProperties` abgefragt werden. Man kann beliebige Anfragen formulieren, da XPath ein sehr mächtiger Tool ist [GrTr 06].

WS-ServiceGroup. Manchmal ist es sinnvoll, aufgrund der gemeinsamen Eigenschaften mehrere Ressourcen zu gruppieren. Dieser Standard fasst zu diesem Zwecke verschiedene Operationen zusammen. Mit Hilfe vom sog. *MembershipContentRules* werden die Gesetze bzw. Kriterien festgelegt, die die Zugehörigkeit zu der entsprechenden Gruppe beeinflussen. Das Ziel der Gruppierung ist im wesentlichen applikationsabhängig. Somit kann man die Zugehörigkeiten nicht nur definieren, sondern auch administrieren. Die Verwaltungs- und Registrierungsfunktionen werden wiederum von einem Web Service „Service Group Registration“ übernommen. Ein Web Service kann mehreren ServiceGroups gehören.

Der Zugriff auf die Gruppen erfolgt durch Referenzen. Jeder Eintrag beinhaltet also eine eindeutige Referenz auf die Ressource und weitere mögliche Metadaten zum Eintrag. Eine ServiceGroup kann nach Bedarf auch Mitgliedschafts-Einschränkungen eintreffen, beispielsweise werden in eine Gruppe nur die Ressourcen aufgenommen, die zu einem bestimmten Web Service gehören. Wie eine ServiceGroup die zugehörige Web Services verwaltet, ist in der Abbildung 2.5 zu sehen. In unserer Arbeit haben wir jedoch keine ServiceGroups definiert, daher werden sie an hier nicht mehr als notwendig betrachtet. Der interessierte Leser sei auf die folgende Quelle [MSB 06] verwiesen.

WS-BaseFaults. In einer SOA-Umgebung ist sehr wichtig, ein vordefiniertes Format für die fehlerbeschreibende Nachrichten zu haben. Diese Spezifikation beinhaltet alle auf XML-Schema basierende Fehlermeldungen. Alle WS-Nachrichten haben eine ähnliche Struktur, da sie alle vom `BaseFaultType` abgeleitet sind. Deswegen beinhalten sie dieselbe Informationen, beispielsweise Fehler-Beschreibung, Zeitstempel, Fehlercode oder Quelle des Fehlers. Durch ein standardisiertes Fehlerformat wird das Management von Fehlern, Erkennung von aufgetrettenen Problemen erleichtert [LiMe 06].

WS-Notification und -Addressing, genannt im selben Kontext, gehören nicht dem WSRF-Standard an. In der Abbildung 2.4 werden sie als non-WSRF Standards dargestellt:

WS-Addressing. Diese Spezifikation beschreibt die Möglichkeit, WS-Resourcen zu adressieren. Dabei

werden die Adressbestandteile definiert. Die Adresse einer WS-Resource besteht aus der WS-Adresse (meistens URI) und der ResourceID, die in der Laufzeitumgebung eindeutig sein muss [GHR 06].

WS-Notification gibt die Möglichkeit so zu konfigurieren, dass jeder Web Service als ein *notification producer* und der Client als ein *notification consumer* agieren. Bei jeder Änderung, sei der Wert einer Ressourceneigenschaft, Änderung eines Zustandes oder der Umgebung, werden die angemeldete Mitglieder benachrichtigt.

Am Anfang war diese Spezifikation Bestandteil vom WSRF, zur Zeit ist sie aber als eigener Standard ausgelagert. Der Mechanismus der Benachrichtigung ist asynchron, somit werden die Messages nur in eine Richtung verschickt. Jeder Teilnehmer, der sich für den Empfang von Nachrichten beim Auftretten eines bestimmten Events registriert hat, bekommt im Falle des Auftrettens eine Kopie der information.

In der Abbildung 2.4 sind die oben beschriebene Spezifikationen, deren WSRF oder non-WSRF Zugehörigkeiten abgebildet.

Mit Hilfe vom WSRF können zustandsbehaftete Ressourcen implementiert werden. Grid Services können also als Web Services aufgefasst werden, wobei die Zustände bei den Ressourcen gespeichert werden. Da ein Grid Service standardisierte Interfaces „nach Außen“ besitzt, kann es deswegen viel einfacher gefunden, aufgerufen und verwendet werden. Die Eindeutigkeit einer Ressource in Grid wird mit Hilfe vom *WS-Resource*-Konzept erreicht. Im Folgenden werden wir kurz die WS-Resourcen und deren Merkmale kurz erläutern [GKM+ 06].

Eine **WS-Resource** besteht aus zwei Komponenten, aus einem *Web Service* und einer *zustandsbehafteten Ressource*. Letztere ist dabei kein Web Service, sondern eine Datenquelle, die Methoden anbietet, um Daten und Zustände zu speichern. Dem Web Service ist die Möglichkeit gegeben, auf die Ressource zuzugreifen. Die Ressource kann ihre Daten in eine Datenbank-Tabelle ablegen.

Der Web Service kommuniziert mit der Ressource nicht durch WS-Nachrichten, da die Ressource kein Web-Service ist. Auf die Ressource greift ausschließlich nur der Web Service zu; er holt alle Daten bezüglich der Zuständen und gibt sie dann weiter „nach Außen“. Wie das Interface „nach Außen“ aussieht, bestimmt das WSRF [Tim 05].

Die Service-Anfrage kann entweder von einem Benutzer oder einem anderen Web Service zu dem entsprechenden Web Service ankommen. Diese Kommunikation ist natürlich eine Web Service Kommunikation, denn es gibt keine direkte Verbindung zu den Ressourcen, nur durch den Web Service.

In der Abbildung 2.6 wird die WS-Resource ausführlich dargestellt. Noch zu beachten ist, dass die zustandsbehaftete Ressource als Abbildung mehrerer Instanzen vorhanden sein kann, wobei alle Instanzen dieselbe Arten von Daten oder Zuständen haben, und sich nur in deren Werten unterscheiden. Dabei hat der zustandsloser Web Service den Zugang (Interface), je nach Bedarf „vom Außen“, zu allen Ressourcen [Tim 05].

Damit jede Ressource eindeutig angesprochen wird, erfordert das Framework eine eindeutige Definition jeder Ressource, durch das Erstellen vom Resource Property Document (RPD), siehe Abschnitt 2.3.

Welche Daten bzw. Zustände aus dem RPD aber „nach Außen“ angeboten werden, wird in der Web Services Description Language (WSDL)-Beschreibung spezifiziert [GrTr 06]. Im folgenden Abschnitt werden wir die Arten von Eigenschaften einer WS-Resource, sowie die Nachrichten-Formate der auf deren Properties angewandten WSRF-Operationen, näher kennenlernen.

2.3 WS-Resource Properties

Der aktuelle Stand jeder WS-Resource wird im Allgemeinen von sogenannten *Eigenschaften* [GKM+ 06] beschrieben. Durch diese werden folgende Arten von Informationen übermittelt:

Service data values sind Informationen über den eigentlichen Zustand der Ressource, wie Ergebnisse einer durchgeführten Operation, Zwischenergebnisse, Laufzeitinformation usw. (Properties wie Personen, Organisationen etc.).

Value metadata sind Zusatzinformationen, die die einzelne Eigenschaften betreffen, z.B. eine Property `LastModifiedBy` gibt den Namen der Person an, die zuletzt den Wert der Eigenschaft gesetzt hat;

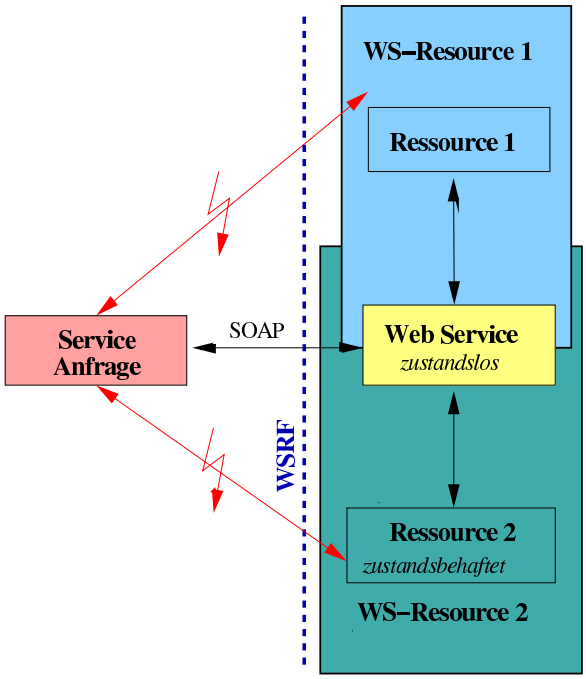


Abbildung 2.6: Übersicht WS-Resource

oder `LastModifiedAt` vermittelt Informationen über das letzte Änderungsdatum der entsprechenden Eigenschaft [Dan 06].

Management Metadaten sind Informationen, die zum Management von Zuständen benötigt werden (ähnlich Metadaten). Sie beziehen sich aber nicht auf die einzelne Eigenschaften, sondern auf die Ressource selbst. Beispielsweise `TerminationTime`-Property spezifiziert das Datum des „Zerstörens“ der Ressource [SoCh 06].

Jede WS-Resource wird in sog. *Resource Property Document* durch die Eigenschaften beschrieben. In der Web Service-Beschreibung (ein WSDL-Dokument) wird das RPD referenziert, dadurch können die Web Services die dazugehörige Ressourcen ansprechen. Auch das RPD ist ein XML-basiertes Dokument [GrTr 06].

Der WSRF-Standard definiert bestimmte Anforderungen bezüglich Resource Properties (RPs). Beispielsweise muss jede WS-Resource ein Resource Property Document, in dem alle Eigenschaften der Ressource eingetragen sind, besitzen. Im RPD muss mindestens die *ResourceID*, eine eindeutige ID der Ressource, stehen. Die weitere ressourcenspezifische Attribute sind optional [Tim 05], trotzdem von den WSRF-Entwicklern empfohlen.

Die WS-ResourceProperty-Spezifikation standardisiert also verschiedene Möglichkeiten, um die Eigenschaften einer Ressource zu formulieren, abzufragen und zu modifizieren. Der Eigenschaften-Aufruf wird durch die angebotene Interface-Operationen erfolgen. Diese bieten mehrere Zugriffsmöglichkeiten: um auf den Wert einer Eigenschaft oder auf die Werte mehrerer Eigenschaften gleichzeitig zuzugreifen. Damit die Anzahl der Kommunikationsnachrichten gering bleibt, wird eine Operation angeboten, die den Zugriff auf das ganze RPD ermöglicht [GrTr 06]. Die WSRF-Spezifikation kann als eine Erweiterung von der RP-Spezifikation von OGSI [TCF+ 03] gesehen werden.

Im letzten Abschnitt haben wir bereits die Stubs angesprochen. Auf der Client Seite wird der Stub, auf dem die Methoden der WS-Resource ausgeführt werden können, PortType genannt. Im Weiteren werden wir den WSRF-ResourceProperty-PortType kurz erklären.

Die WSRF-Spezifikation (bzgl. WS-ResourceProperties) definiert folgende PortType-Operationen [GrTr 06]:

GetResourceProperty. Durch diese Operation wird ermöglicht, die einzelne Eigenschaftswerte einer Ressource zu lesen. Zusätzlich wird ein `QName` für jede Eigenschaft definiert. Somit müssen die Entwickler

keine individuelle Operationen für den Zugriff auf jede Property selber schreiben.

Jede WS-Resource soll GetResourceProperty `Request Messages` eines Requestors im folgenden Format akzeptieren (Listing 2.1):

Listing 2.1: Nachrichten-Format eines GetResourceProperty-Requests [GrTr 06]

```
<wsrf-rp:GetResourceProperty />
    QName
</wsrf-rp:GetResourceProperty>
```

Bei jeder Anfrage der Operation muss der *Qualified Name* (QName) mit angegeben werden, der einzelne Objekte innerhalb des Web Services referenziert. Der QName kann auch ausserhalb vom Client zum Referenzieren verwendet werden. Strukturell besteht er aus zwei Teilen: aus einem geklammerten Namespace und aus einer lokalen Variable (z.B. {`http://www.examples.org/impl`}`Status`)

Das Antwort-Format auf die Request-Nachricht sieht wie folgt aus 2.2:

Listing 2.2: Nachrichten-Format einer GetResourceProperty-Response [GrTr 06]

```
 <wsrf-rp:GetResourcePropertyResponse>
  {any}*
   </wsrf-rp:GetResourcePropertyResponse>
```

Falls die WS-Resource auf die GetResourceProperty-Anfrage nicht mit einer GetResourcePropertyResponse antwortet, muss eine Fehler-Meldung zurückgeschickt werden. Diese wird in der WS-BaseFaults-Teilspezifikation spezifiziert.

GetMultipleResourceProperties. Beim Aufruf dieser Methode werden gleichzeitig mehrere Eigenschaften einer Ressource angezeigt. In welchem Format die Requests und Responses vom Client an die WS-Resource geschickt werden sollen, wird im WSRF, wie in den folgenden Listings 2.3 und 2.4, spezifiziert.

Listing 2.3: Nachrichten-Format eines GetMultipleResourceProperties-Requests [GrTr 06]

```
<wsrf-rp:GetMultipleResourceProperties>
    <wsr/f-rp:ResourceProperty>QName<wsrf-rp:ResourceProperty>+
</wsrf-rp:GetMultipleResourceProperties>
```

Listing 2.4: Nachrichten-Format einer GetMultipleResourceProperties-Response [GrTr 06]

```
<wsrf-rp:GetMultipleResourcePropertiesResponse>
    {any}*
</wsrf-rp:GetMultipleResourcePropertiesResponse>
```

SetResourceProperties. Die Properties einer Ressource können nicht nur gelesen, sondern auch gesetzt werden. Diese Operation bietet mehrere Änderungsmöglichkeiten, wie `update` (der aktuelle Property-Wert wird durch einen neuen ersetzt), `delete` (der Wert wird gelöscht) oder `insert` (eine neue Property mit einem neuen vorgegebenen Wert wird hinzugefügt). Die Exchange-Nachrichten müssen den folgenden Formaten wie in den Listings 2.5 und 2.6 entsprechen.

Listing 2.5: Nachrichten-Format eines SetResourceProperties-Requests [GrTr 06]

```
<wsrf-rp:SetResourceProperties>
[
  <wsrf-rp:Insert >
      {any}*
           </wsrf-rp:Insert> |
     <wsrf-rp:Update >
     {any}*
   </wsrf-rp:Update> |
<wsrf-rp:Delete ResourceProperty="QName" />
```

```
]+
</wsrf-rp:SetResourceProperties>
```

Listing 2.6: Nachrichten-Format einer SetResourceProperties-Response [GrTr 06]

```
<wsrf-rp:SetResourcePropertiesResponse>
</wsrf-rp:SetResourcePropertiesResponse>
```

Die Formate der weiteren `Exchange Messages`, wie bei der UpdateResourceProperties- oder der InsertResourceProperties-Operationen, ähneln sich der oberen Darstellung. Deswegen werden sie nicht ausführlich hingeschrieben.

QueryResourceProperties. Diese Operation bietet die Möglichkeit komplexe Anfragen an die Ressourcen bzgl. Eigenschaften zu stellen. Die benutzte Adressierung dafür ist die XPath [4] (XML-Anfrage). Die Formate für die `Exchange Messages` sind wie in den Listings 2.7 und 2.2 vorgeschrieben.

Listing 2.7: Nachrichten-Format eines QueryResourceProperties-Requests [GrTr 06]

```
<wsrf-rp:QueryResourceProperties>
    <wsrf-rp:QueryExpression Dialect="xsd:any"/>
</wsrf-rp:QueryExpression>
```

Listing 2.8: Nachrichten-Format einer QueryResourceProperties-Response [GrTr 06]

```
<wsrf-rp:QueryResourcePropertiesResponse>
    {any}
</wsrf-rp:QueryResourcePropertiesResponse>
```

GetResourcePropertyDocument. Wollen wir alle Eigenschaften einer bestimmten Ressource lesen, wird diese Methode aufgerufen. Das reduziert den Kommunikationsaufwand im Vergleich zum Aufruf der einzelnen Eigenschaften. Die Formate der Anfrage- und Antwort-Nachrichten sind in dem WSRF-Standard, wie in den folgenden Listings 2.9 und 2.10 vorgeschrieben.

Listing 2.9: Nachrichten-Format eines GetResourcePropertyDocument-Requests [GrTr 06]

```
<wsrf-rp:GetResourcePropertyDocument />
```

Listing 2.10: Nachrichten-Format einer GetResourcePropertyDocument-Response [GrTr 06]

```
<wsrf-rp:GetResourcePropertyDocumentResponse>
    {any}
</wsrf-rp:GetResourcePropertyDocumentResponse>
```

PutResourcePropertyDocument. Mit Hilfe dieser Funktion kann jeder Requestor die Werte aller Eigenschaften eines RPDs durch neue ersetzen. Die Formate für die PutResourcePropertyDocument-Nachrichten ähneln sich den GetResourcePropertyDocument-Messages.

InsertResourceProperties. Diese WSRF-Operation bietet die Möglichkeit, eine oder mehrere Ressource-Eigenschaften zu der entsprechenden Ressource hinzufügen.

UpdateResourceProperties. Diese Operation ermöglicht das Ersetzen von aktuellen Werten einer Eigenschaft durch die anderen.

DeleteResourceProperties. Um eine oder mehrere Eigenschaften einer Ressource zu löschen, wird die Funktion DeleteResourceProperties aufgerufen.

In der WSDL-Datei des Web Services müssen wir die RPss und deren Funktionen, die wir verwenden wollen, spezifizieren. Wie die Definition von Eigenschaften einer Ressource in WSDL (Version 1.1) eingebettet werden können, wird in dem folgenden Listing 2.11 beschrieben.

[4] Siehe auch http://www.w3.org/XML/Linking

Listing 2.11: RP-Definition in WSDL [GrTr 06]

```
<wsdl:defintions ...>
    <wsdl:portType ...
        wsrf-rp:ResourceProperties="xsd:QName"? ... >
        ...
    </wsdl:portType>
</wsdl:definitions>
```

Die Einbindung von zu verwendenden RP-Operationen aus dem WSRF-Standard in der WSDL-Datei erfolgt wie im folgenden Listing 2.12.

Listing 2.12: RP-Operationen in WSDL [GrTr 06]

```
<portType name="VOPortType"
    wsdlpp:extends="wsrpw:GetResourceProperty
     wsrpw:SetResourceProperties
     wsrpw:GetResourcePropertyDocument
     wsrpw:PutResourcePropertyDocument
     wsrpw:InsertResourceProperties
     wsrpw:DeleteResourceProperties
     wsrpw:GetMultipleResourceProperty
     wsrpw:QueryResourceProperties"
     wsrp:ResourceProperties="tns:VOResourceProperties">
```

Nachdem wir nun die Spezifikationen des WSRF, und im Besonderen die Definition von WS-ResourceProperties betrachtet haben, benötigen wir für die Umsetzung unserer VO als WS-Resource noch einen Rahmen, der das WSRF implementiert. Dazu bedienen wir uns des Globus Toolkits, das in der Version 4 nun im Kern auf dem WSRF aufbaut.

2.4 Globus Toolkit 4

Das Globus Toolkit 4 (GT4)[5] ist ein Vermittlungslayer zwischen der Anwendungs- und der Systemschicht. GT4 ist eine Technologie zur Erstellung von Grids, auf dem ein Anwender seine Applikationen laufen lassen kann und wird vom Globus Alliance entwickelt. Die Hauptaufgabe vom Globus ist die heterogene, weltweit verteilte Ressourcen zu koordinieren, und sie wirtschaftlich und sicher zu nutzen [SoCh 06].

Die GT4 Middleware stellt Software für die Sicherheit, das Data-, Ressourcen- und Informations-Management, die Portabilität und Fehlererkennung bereit. Dabei implementiert sie und basiert auf den Spezifikationen des in den vorangegangenen Abschnitten beschriebenen WSRF.

Die wichtigsten Module von GT4 sind in Abbildung 2.7 zu finden. Einige dieser Komponenten zählen zu den WS-Komponenten, andere gehören zu der non-WS-Komponenten-Gruppe. Die nicht-WS-Module bilden die GT4-Basis und stellen die Grid-Services den WS-Komponenten zum Zugriff bereit. Farblich sind die abgebildete Module, die zu verschiedenen Bereichen gehören, abgetrennt. Im Folgenden werden wir jede dieser Komponenten näher betrachten:

Security besteht aus mehreren Teilkomponenten, die wichtigsten aber dabei ist Grid Security Infrastructure (GSI) [SoCh 06], da sie durch die eingebaute Security-Mechanismen die Realisierung von entfernten Grids erlaubt. Mit Hilfe von diesem Werkzeug ist es möglich eine Umgebung zu schaffen, so dass jeder Anwender sicher seine Aufträge auf die Remote Computer ausführen kann [Korn 04].

Data Management bietet den Transfer und Zugang von bzw. zu den Daten [SoCh 06].

Execution Management nimmt die Grid-Aufträge entgegen und bearbeitet sie [SoCh 06].

Information Services stellen Teilkomponenten zur Entdeckung und Überwachung von Ressourcen in VOs [SoCh 06].

[5] Ab Juni 2007 GT Version 4.0.5 verfügbar. Weitere Informationen unter http:/http:///www.globus.org/toolkit/

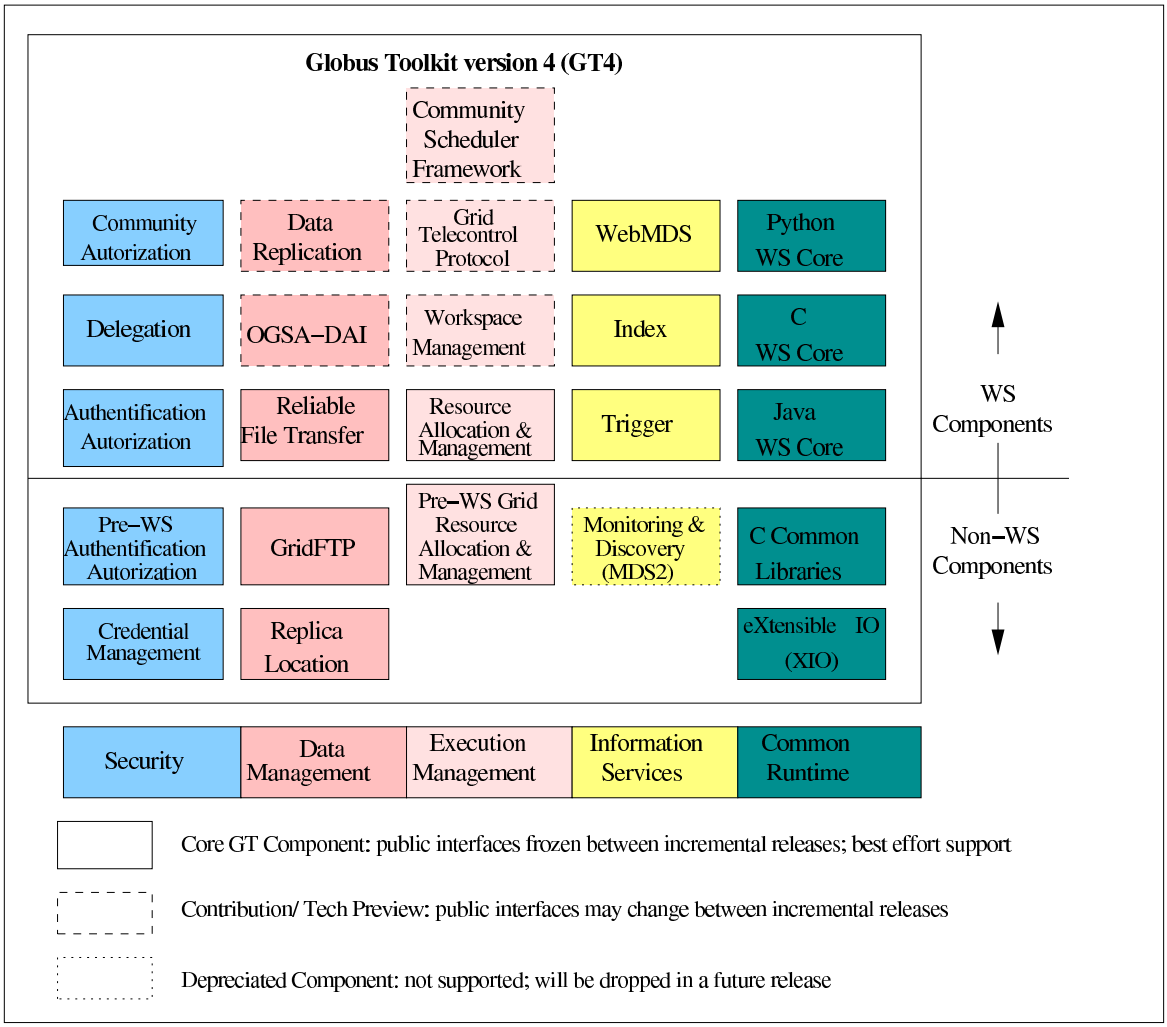


Abbildung 2.7: Komponenten vom Globus Toolkit 4 [SoCh 06]

Common Runtime. Mit Hilfe der Application Programming Interface (API)-Schnittstellen, die es für mehrere Programmiersprachen, wie C, Java, Python, Perl etc. gibt, lassen sich Anwendungen für den Grid-Einsatz anpassen. Die *Common Runtime* beinhaltet Werkzeuge und Bibliotheken, die für den Entwickler verschiedener Programmiersprachen zur Hilfe kommen. Wie in der Abbildung schon gezeichnet, ändert sich die Anzahl bzw. Übersicht von Komponenten nach jeder Version [SoCh 06].

Alle oben beschriebene Module bieten einem Client die Möglichkeit, auf die Ressourcen in einem Grid zuzugreigen. Der Begriff des Grid Computings, sowie die Grid-Architektur werden im Folgenden dargestellt.

2.5 Grid Computing

Der Begriff *Grid Computing* wurde am Ende der 90er Jahren [BHF 03] zum ersten Mal eingeführt und damals nur bei den Wissenschaftlern richtig bekannt. Heutzutage werden Grids nicht nur für die Forschung, sondern auch für kommerzielle Zwecke eingesetzt.

Sehr oft werden die Begriffe *Grid* und *Cluster* verwechselt. Es liegt an der Natur von diesen Konzepten, da sie beiden ein und dasgleiche Ziel verfolgen, nämlich die ganze Rechenkraft von allen verfügbaren Ressourcen zu einer Hochleistungs-Ressource nutzbar zu machen [IBM 07].

Der grosse Unterschied zwischen einem Grid und einem Cluster ist, dass ein Grid heterogen aufgebaut ist,

wobei die Ressourcen weltweit verteilt sind. Die Ressourcen, die einen Cluster bilden, befinden sich aber an einem geographischen Ort und haben als Basis gleiche Software- und Hardware-Systeme.

Während der Cluster ein Verbund mehrerer Rechnern darstellt und die Kommunikation über ein leistungsfähiges Local Area Network (LAN) erfolgt, fasst das Grid mehrere Ressourcen zusammen, die untereinander unter Wide Area Network (WAN) kommunizieren.

Außerdem wird ein Cluster von einem zentralen Administrator administriert, ein Grid-System dagegen besteht aus Ressourcen, die je eigene Administration besitzen. Alle Teilnehmer einer Virtuellen Organisation (Definition und Charakteristika unter 2.6 zu finden) haben zwar ein gemeinsames Ziel, aber keine gemeinsame Administrationsplattform, da die Ressourcen unterschiedlichen Providern[6] gehören [Prof 05].

Zu einer Grid-Infrastruktur gehören nicht nur Hochleistungsrechner und Speicher, sondern auch kostspielige Teleskope, Mikroskope, Netzwerke, Datenbanken, Drucker, beliebige Software usw. Eine weitere Aufgabe ist die Integration, Auswertung und Darstellung von riesen Forschungs-Datenmengen.

In der Literatur gibt es unzählige Grid-Definitionen. Eine davon ist:

> *„Ein Grid koordiniert Ressourcen, die nicht einer zentralen Instanz untergeordnet sind und verwendet offene, standardisierte Protokolle und Schnittstellen, um nicht triviale Dienstgüten bereitzustellen.“* [Fost 02]

Ein Grid koordiniert also Ressourcen, die in der Regel nicht einer einzigen Organisation gehören und geografisch weltweit verteil sind; benutzt verschiedene offene Protokolle, die Autorisierung, Authentifizierung usw. regeln; alle Ressourcen, die verwendet werden, koordinieren miteinander, damit verschiedene Dienstgüte-Anforderungen erfüllt werden können.

Foster und Kesselmann geben eine andere Definition, die verschiedene Grid-Grundmerkmale spiegelt:

> *„A grid is a hardware and software infrastructure that provides dependables, consistent, pervasive, inexpensive access to high end computational capabilities.“* [FoKe 01]

Mit „dependable“ wird die Zuverlässigkeit (Garantie für Funktionalität) gemeint; „consistent“ heisst, dass überall, egal an welchem Ort und bei welchen Organisationen, dieselben Dienste bereitgestellt werden müssen. „Pervasive“ heisst, dass die angebotene Grid-Dienste von überall aus erreichbar sein müssen. „Inexpensive“ bezieht sich auf die Kosten und bedeutet, dass Kosten nicht zu hoch sein dürfen.

Im Folgenden werden wir die Grid-Architektur [FoKe 01] kurz darstellen.

Die Abbildung 2.8 zeigt uns die Schichten, die in einer Grid-Architektur vorhanden sind, sowie die Ähnlichkeiten der Grid- zur Open Systems Interconnection (OSI)-Architektur [FoKe 01]. Da der Begriff vom Grid Computing hier nur kurz beschrieben wird, wird die OSI-Architektur nicht dargestellt. Der interessierte Leser sei aber auf die folgende Quelle [Zimm 80] verwiesen.

Fabric Layer. Hier werden alle am Grid angemeldeten Ressourcen (z.B. Speicherressourcen, Hochleistungsrechner, Teleskope, Datenbanken, Netzsegmente etc.) für den Zugriff von Protokollen bereitgestellt. Alle für die Ressourcen spezifische Operationen werden hier implementiert; durch eine festgelegte Schnittstelle werden dann diese Ressourcen von den höheren Schichten angesprochen.

Connectivity Layer. Auf diesem Layer werden spezifische Kommunikations- und Authentifizierungsprotokolle (z.B. für Sicherheit) definiert. Somit können die Fabrik-Ressourcen miteinander kommunizieren. Auf die Kommunikationsprotokolle bauen dann die Authentifizierungsprotokolle (APs) auf. Die wichtigsten Internet-Protokolle wie Transmission Control Protocol (TCP), Internet Protocol (IP), Hypertext Transfer Protocol (HTTP), Domain Name System (DNS) etc. gehören zu dieser Schicht.

Resource Layer. Diese Schicht baut auf den Protokollen des unteren Connectivity-Layers auf und sorgt für eine sichere Übertragung, Initiierung, Monitoring, Accounting etc. für gemeinsame Operationen auf individuellen Ressourcen. Diese Schicht kommuniziert mit der Fabrik-Schicht, um die Ressourcen zu verwenden und zu kontrollieren. Zwei Arten von Protokollen sind hier zu unterscheiden [RöSc 05]:

Informationsprotokolle werden eingesetzt, um aktuelle Informationen (Konfiguration, Anzahl von laufenden Prozessen etc.) über Ressourcen zu erfahren.

[6] Oberbegriff für die Anbieter von Ressourcen, Anwendungen und Diensten in einem Grid

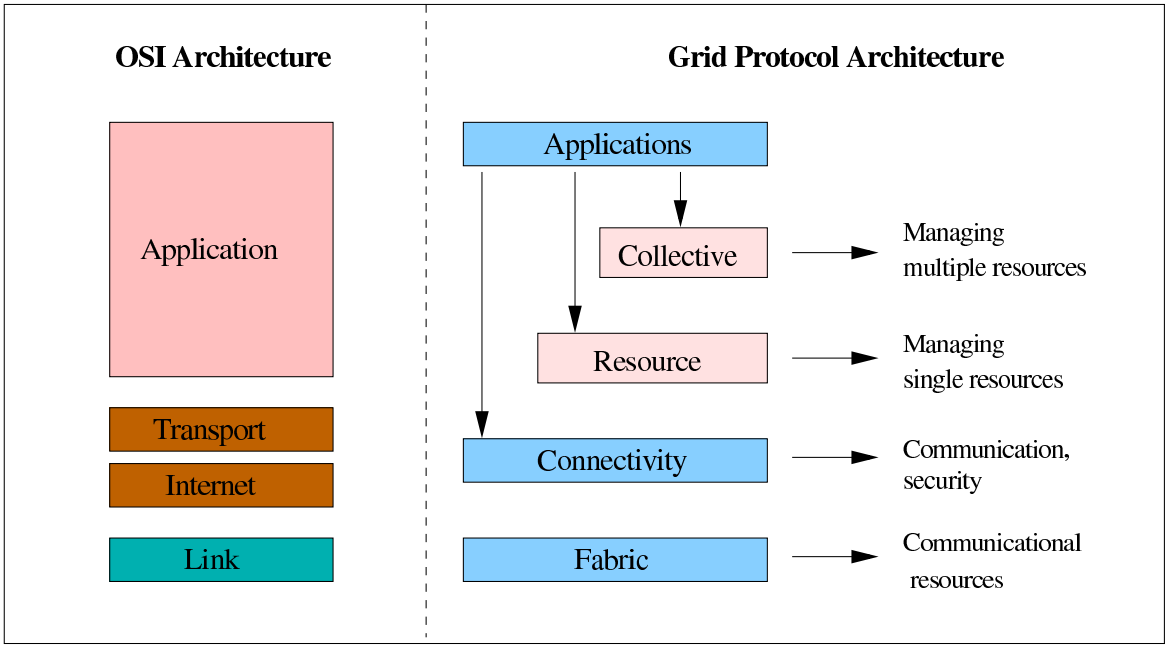


Abbildung 2.8: Grid Architektur [Sess 03]

Managementprotokolle werden bei der Überwachung, Datenkontrolle etc. von Ressourcen benötigt.

Collective Layer. Die Schicht beschreibt, wie die einzelne Ressourcen miteinander interagieren. Dafür werden mehrere Collective-Dienste für Monitoring und Diagnose, Datenreplizierung, Gemeinschaftsbuchung und -zahlung, Scheduling etc. angeboten.

Application Layer. Hier sind die Benutzer-Applikationen einer Community (VO) anwesend.

Die Services, die für Grid Computing entwickelt wurden, heissen „Grid Services“. Diese sind nicht anders als klassische Web Services (siehe auch Unterkapitel 2.1), die um Zustandsspeicherung erweitert worden sind. Bei den Grid-Aufgabenstellungen handelt es sich meistens um sehr komplizierte und komplexe Aufgaben, wo die Zwischenergebnisse sehr wichtig für das weitere Rechnen sind. Daher müssen die Grid Services *zustandsbehaftet* sein. Im Weiteren werden wichtige *Anforderungen* an Grid Services aufgelistet:

- „kurzlebig“. Die Verwaltung von Grid Services verläuft über die Instanzen. Wird der Service (die Instanz) nicht mehr benötigt, da die Berechnung erfolgreich durchgeführt wurde, wird die Instanz zerstört, im Vergleich zu den Web Services, die von Natur aus persistent sind.
- „zustandsbehaftet“. Ein anderes Merkmal des Web Services ist, dass er zustandslos ist. Komplexe Grid-Aufgaben fordern zustandsbehaftete Services an, da bei solchen Berechnungen mehrere Zustände erreicht werden können. Also soll jeder Grid Service zustandsbehaftet sein.
- „eindeutige Adressierung“. Da durch die Instanz-Erzeugung ein Web Service mehrfach vorkommen kann, muss jeder Grid Service eindeutig adressiert werden.
- „Lebenszyklusmanagement“. Jeder Grid Service wird durch unterschiedliche Maßnahmen (Management) geregelt, die die Verwaltung von Instanzen ermöglichen.

Um alle diese Anforderungen zu erfüllen und zu standardisieren, wurden mehrere Standards entwickelt, wie Open Grid Services Architecture (OGSA) [FKS+ 06], Open Grid Services Infrastructure (OGSI) [TCF+ 03], Web Services Resource Framework (WSRF) [Tim 05] (Siehe auch Abschnitt 2.2).

Eine Vielzahl von Projekten zeigt, dass das Grid-Computing sowie in der Wissenschaft, als auch in der Wirtschaft Alltag gefunden hat. Interessant dabei ist, nicht nur die Bearbeitung einzelner Aufgaben, sondern vielmehr die Abbildung komplexer Prozesse, die durch eine variable Anzahl von Personen und Ressourcen gekennzeichnet sind. Durch seine Struktur bietet das Grid die Möglichkeit an, solche Prozesse zu unterstützen.

Für die Strukturierung dieser Prozesse werden Virtuelle Organisationen gebildet.

2.6 Virtuelle Organisation

Die Definition einer „Virtuellen Organisation (VO)“ erschien zum ersten Mal im Jahr 1986 im angloamerikanischen Raum [Abbe 97]. Seitem entstanden unterschiedliche Definitionen. Bis jetzt hat sich aber keine allgemeingültige Definition durchgesetzt [SZM 06a].

Sehr viele Web-Dokumente zitieren die Definition von Foster und Kesselman, allerdings weist diese keine Dynamik in einer VO nach:

> *„A set of the individuals and/ or institutions defined by resource sharing rules.“* [FoKe 01]

Innerhalb einer VO werden Regeln für gemeinsame Nutzung von Ressourcen und Services festgelegt, d.h. wer, wie, wann auf welche Services oder Ressourcen zugreifen kann oder darf. Einfacher ausgedrückt, geht es bei einer VO um *Ressourcen und Service-Sharing*. Solche Regeln sind dynamisch zu erstellen und der VO zuzuweisen. Jede Ressource, jede Institution kann jederzeit ohne Probleme zu VO hinzugefügt oder aus der VO entfernt werden. Zu jeder VO gehört eine beliebige Gruppierung von Ressourcen, Individuen, Organisationen etc., ohne Rücksicht auf die geografische Eigenschaften zu nehmen. Außerdem wirkt jede VO wie eine „geschlossene Einheit“ gegenüber den anderen. Zu einer Virtuellen Organisation kann auch eine physische Organisation zugehören, indem sie ihre Ressourcen freigibt oder zur Verfügung stellt.

Der Aspekt der gemeinsamen Interessenslage und die dafur erforderlichen Koordinationsmechanismen wird vom Vox Project Team so definiert:

> *„A Virtual Organization consists of members that may come from many different home institutions, may have in common only a general interest or goal (e.g., CMS physics analysis), and may communicate and coordinate their work solely through information technology (hence the term virtual). In addition, individual members and/or institutions may join and leave the organization over time; sometimes VOs are called dynamic virtual organizations for this reason.“* [VOX 04]

Eine VO wird in inter- und intraorganisatorische Gestaltungskontexte [Schi 07] verwendet. Intraorganisatorisch heißt, dass die Virtuelle Organisation das Ziel verfolgt, räumliche und zeitliche Begrenzungen zu überwinden. Als Prinzip der interorganisatorischen Gestaltung unterstützt eine VO die institutionelle Organisationsperspektive. Somit bildet eine VO ein flexibles Netzwerk von Organisationen, die rechtlich selbstständig sind. Die Organisationen benutzen gemeinsam ihre jeweilige Kompetenzen und Ressourcen. In unserer Arbeit wird aber nur die intraorganisationelle VO-Kooperation betrachtet. Der interessierte Leser sei auf die einschlägige Literatur [Müll 97] verwiesen.

Aufgrund der oben zitierten Definitionen können folgende VO-Anwendungsfälle intuitiv abgeleitet werden [SZM 06a]:

- jede VO besteht aus einem oder mehreren Mitgliedern
- die VOs können Mitglieder anderer VOs sein
- eine VO kann als Mitglied eine weitere VO besitzen
- eine VO besteht aus virtuellen, nicht reellen Ressourcen
- eine VO ist keine gesetzlichdefinierten Organisation (legal entity).

Eine VO kann auch aus der Sicht der End-User, Provider, Manager oder Developer wie folgt definiert werden:

- *User*: Der End-User meldet sich bei einer bestimmten VO, mit dem Wunsch deren Dienste und Ressourcen zu benutzen.
- *Service Developer*: Der Service Provider stellt einen oder mehrere Dienste für eine bestimmten Community (VO) bereit, die dann weiterhin gepflegt werden.
- *Resource Provider*: Der Resource Provider stellt Ressourcen und Maschinen, die zum Grid Service Einsatz benötigt werden, bereit.

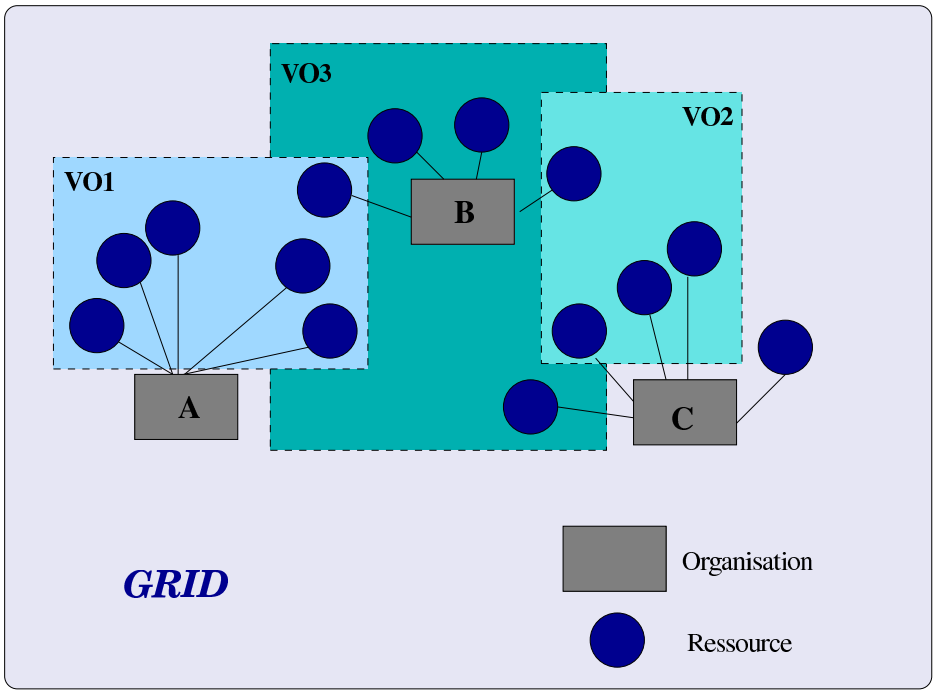


Abbildung 2.9: Bildung Virtueller Organisationen

- *Manager*: Der Manager administriert jede VO und vergibt jedem Mitglied Zugriffsrechte innerhalb einer VO.

Die Bildung Virtueller Organisationen ist natürlich nur dann *sinvoll* wenn es gilt, daß es keine gemeinsame Zentrale für die Beteiligten gibt; wenn keiner von den Parteien volle Kontrolle über die Organisation hat; wenn jeder von den Mitgliedern nicht genügend Kenntnisse oder Ressourcen besitzt und, um das vorgesehene gemeinsame Ziel zu erreichen, Kompetenzen und Ressourcen von den anderen benötigen würde. Aus welchen Komponenten eine VO besteht und wie die Bildung stattfindet, ist es in der Abbildung 2.9 zu erkennen.

Eine Virtuelle Organisation ist für die kundenorietierten Lösungen spezifisch. Für die Massenproduktion sind andere Formen des Unternehmens üblich, da sie besser dafür geeignet sind. Außerdem variieren stark die Grössen von VOs, wegen unterschiedliche Anzahl von Mitgliedern, unterschiedliche Lebensdauer, Strukturen etc. [ScSt 07].

Im Folgenden werden wir einige Eigenschaften einer VO gegenüber Realen Organisationen (RO) aufzählen:

- Wegen der Virtualität der VO ist der *physische Standort* irrelevant, anders als bei der Realen Organisation (siehe [RPW 03]). Die Ressourcen bzw. Mitglieder einer VO sind in der Regel weltweit verteilt. Außerdem gehören sie unterschiedlicher Organisationen bzw. Standorten.
- Die Kommunikation verläuft durch ein Netzwerk, es gibt also keine *direkte Kommunikation* zwischen den Beteiligten. Da in einer VO nur mittels Web Services auf die Ressourcen zugegriffen wird, wird eine Netzwerk-Kommunikation vorausgesetzt.
- Die *Zielsetzung* wird durch Kooperation aller Beteiligten bestimmt (alle Beteiligten eines Forschungsprojektes beabsichtigen dasselbe Ziel).
- Die *Gründung* einer VO erscheint viel *flexibler*, als einer anderen Form der Organisation, da VOs ohne Probleme kooperieren können, ohne dafür ein Unternehmen zu gründen, einen Standort zu finden, Personal einzustellen und eine Organisation aufzubauen. Der Formalisierungsgrad ist also sehr gering. Wie oben schon erwähnt, ist eine VO keine „legal entity“; im Verleich zur Realen Organisation werden die Teilnehmer in einer VO nicht richtig „angestellt“[RPW 03].
- *Dynamik* erscheint nicht nur bei der *Struktur* der VO, sondern auch bei den *internen/ externen Prozessen*.

Jedes Individuum, jede Ressource und jede VO kann sich zu jeder Zeit an die eigentliche VO anschließen und sie nach Bedarf bzw. Wunsch auch verlassen. Die VO befindet sich in ständiger Umstrukturierung [SZM 06b].

- Da die VOs ein bestimmtes Ziel verfolgen, sollen sie nacher in der Regel aufgelöst werden. Die VOs sind im Wesentlichen *zeitlich befristet*, also ist die Dauer der Existenz vordefiniert. Im Kapitel 3 haben wir dazu ein erklärendes VO-Szenario beschrieben. Wenn sich Forscher im Rahmen eines Projektes zu einer VO zusammenschließen und nach einer bestimmten Zeit das definierte Ziel erreichen, werden die gebildeten VOs bzw. die dazugehörige Sub-VOs letztlich wieder aufgelöst.
- Im Vergleich zu den anderen Formen von Unternehmen werden viele *Kosten* bei der Gründung sowie während der Existenz gespart, z.B. Transport-, Organisations-, Raum-, Kommunikations-Kosten, gemeinsame Benutzung von teueren Ressourcen etc. Durch die gemeinsame Nutzung von Ressourcen werden beispielsweise die Kosten der Ressourcen-Nutzung zwischen den Teilnehmern geteilt.
- Die Dynamik bei der VO hat nicht nur Vorteile. Jeder Teilnehmer einer VO, beim Verlassen, Betretten einer VO verursacht *Verwaltungskosten*. Will eine Person beispielsweise eine VO betretten, muss der Administrator entsprechende Rechte freigeben.
- *Autorisierung* ist auch ein wichtiger Aspekt, das während der Existenz einer VO ständig unter Kontrolle gesetzt wird. Jedem Midglied werden z.B. verschiedene (eingeschränkte) Rechte (Zugriff, Verwendung etc.) vergeben, die sich im Laufe der Zeit ändern können [RöSc 05].

Aus der **Management-Sicht** [7] werden solche VOs durch Dimensionen, die in der Abbildung 1.2 dargestellt sind, beschrieben [SZM 06b].

So kann die *Lebensdauer* einer VO sowohl kurzfristig, mittelfristig, als auch langfristig sein. Der *Lebenszyklus* erstreckt sich dabei auf mehrere Phasen, von der Planung bis zur Auflösung der VO, genauso wie bei den physischen Unternehmen.

Vor dem *Gründungsprozeß* jeder VO wird auch entschieden, ob die Organisation geplant, spontan oder bei Bedarf virtuell gegründet wird. Mehrere *VO-Strukturen* können gebildet werden, offen, halb-offen oder geschlossen.

Das *Management* einer VO bezieht sich auf Inter-Grid Prozesse, Inter-VO Prozesse und VO-interne Prozesse. Außerdem kann die *Mitglieder*-Anzahl einer VO sowohl statisch, dynamisch und bekannt, als auch dynamisch und unbekannt sein.

Die gebildeten *Kooperationen* können unterschiedliche Strukturen haben, wie Stern, P2P oder Lieferkette. Ein anderes Charakteristikum ist die *Administration*, die entweder manuell, werkzeugunterstützt oder automatisiert verläuft.

In der Abbildung 1.2 wird außerdem der Fokus aktueller VO Management-Ansätze dargestellt. Wie abgebildet, werden viele Eigenschaften einer VO entweder teilweise oder gar nicht in Betracht gezogen; unzureichend erforscht sind z.B. Bereiche, wie die Betrachtung kurzlebiger VOs, Unterstützung von offenen Gruppenstrukturen usw. [Schi 07].

Wie oben schon erwähnt, betreffen die Management-Maßnahmen in einer VO einerseits die VO-internen Prozesse, wobei VO-Bildung vorausgesetzt wird, andererseits wird der Betrieb einer VO als *managed object* gesteuert. Einer der wichtigsten VO-internen Prozessen ist das Mitgliedschafts-Management.

In der Abbildung 2.10 sind ein Paar Anwendungsbeispiele für die beiden VO Management-Arten aufgelistet. Aus der Sicht der *managed objects* wird das Lebenszyklus der VO in Vordergrund gesetzt und aus der Sicht der internen Prozessen werden die Maßnahmen auf die Dienste und Ressourcen fokusiert.

Das **Management von Mitgliedschaften** in einer VO bezieht sich auf die Maßnahmen, die VO-Mitgliedschaft regelt. Jedem Teilnehmer einer VO werden Zugriffsrechte auf Ressourcen und/ oder Services zugewiesen. Eine Anforderung dafür wäre die Realisierung einer adequaten Autorisierungs- und Authetisierungsinfrastruktur (AAI), d.h. *Identitätsmanagement*.

[7] Das Management einer VO bezeichnet die Gesamtheit von Maßnahmen (Best Practice) und Methoden, die nötig sind, um die bestmögliche Unterstützung-effektiv und effizient- von Geschäftsprozessen einer VO zu erreichen.

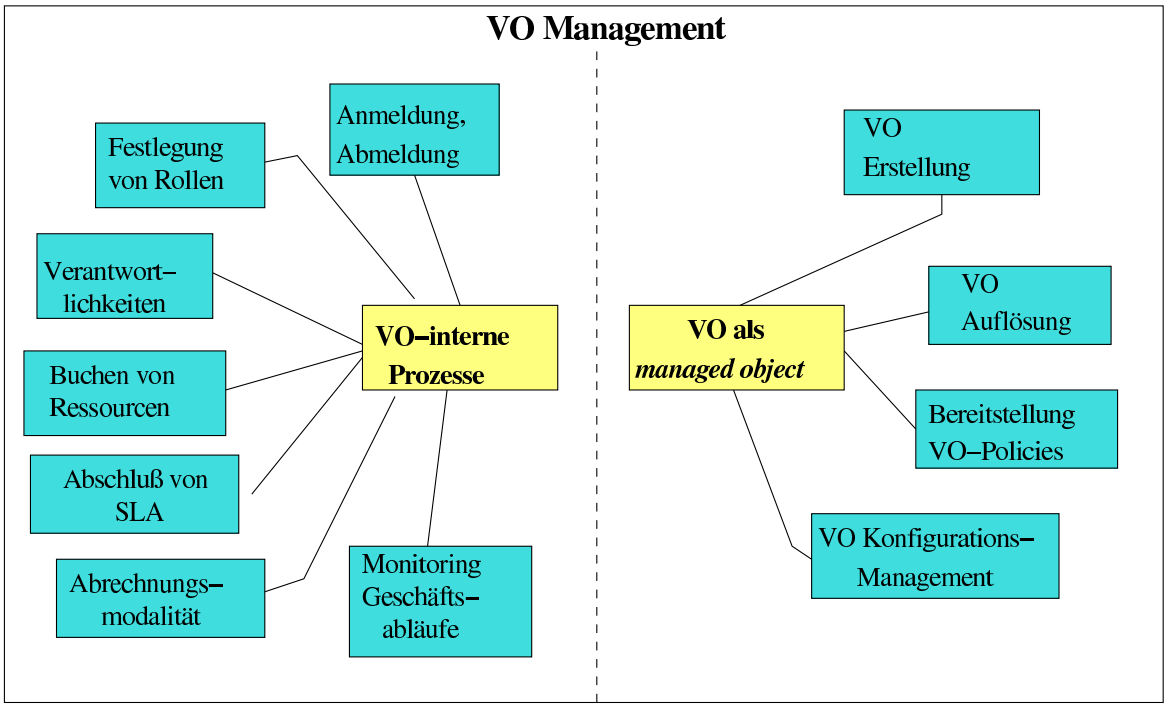


Abbildung 2.10: VO Management Szenarien

Authentifizierung ist der Nachweis einer Identität; *Autorisierung* beruht auf die Zuweisung und Überprüfung von Zugriffsrechtenn auf die Ressourcen/ Services. Die Zugriffsrechte eines VO-Teilnehmers werden durch sog. *Attribute* beschrieben. Das Hauptziel dieses Managementes ist diese Attribute für jeden Teilnehmer zu pflegen und die entsprechende Provider über die Rechte zu informieren [SZM 06b].

Beim **Lebenszyklusmanagement** werden die Virtuellen Organisationen als *managed objects* analysiert. Es wurden viele Managementmodelle entwickelt, aber keins davon wird genügend spezifiziert [Schi 07]. Unterschiedliche Services werden für jede der Lebenszyklusphasen vom Management zur Verfügung gestellt. Das Lebenszyklus – von der Planung bis zur Auflösung – wird in der Tabelle 2.1 dargestellt.

Planungsphase	Die VOs werden auf Basis von sog. *templates* erzeugt, wobei bestimmte Initialparameter vordefiniert sind, wie Dienstgüterparameter, Verlässlichkeitskriterien, Listen der Benutzer, Rollenabbildung von Benutzern, Provider-Präferenzen, VO-Policies etc.
Formationsphase	In dieser Phase wird eine VO (noch keine Mitglieder beinhaltend) vom VO-Provider (Factory Service) initialisiert, der die Service Level Agreement (SLA) [HAN 99]-Verhandlungen mit den Anbietern von Ressourcen und Services übernimmt. Die entsprechende Provider werden zu den VOs hinzugefügt.
Betriebsphase	In dieser Phase werden die VO-, und mit ihr der Ressourcen-Lebenszyklus synchronisiert, gefolgt von Bereitstellung von klassischen FCAPS-Netzwerkmanagement-Diensten (*Fehler, Konfiguration, Abrechnung, Leistung, Sicherheit*) [HAN 99]
Anpassungsphase	In dieser Phase wird aufgrund der verursachten Fehler oder Veränderungen das System angepasst (z.B. neues Hinzufügen/ Entfernen von Ressourcen, Diensten oder Providern etc.)
Auflösungsphase	Wenn das vordefinierte Ziel einer VO erreicht wurde, wird diese letztendlich zerstört.

Tabelle 2.1: Lebenszyklusmanagement einer VO [SZM 06b]

Die **Management-Anforderung** an eine Virtuelle Organisation betreffen das Erstellen, das Aktualisieren von Informationen, die Überwachung und die Auflösung. Bei unserer Implementation machen wir uns die Tatsache zunutze, dass bereits Werkzeuge zum Managen von WS-Resourcen existieren. Mit diesen soll nun auch eine

Virtuelle Organisation verwaltet werden können, daher die Umsetzung gemäß den entsprechenden Standards.

Einem zentralen Management fallen verschiedene Verwaltungstätigkeiten zu. Außerdem ist es die Anlaufstelle für die verschiedenen administrativen Anforderungen der Mitglieder der Organisation. Hierbei ist es wichtig, verfahrensweisen und Berechtigungen für Anforderungen, sowie Prozeßbeschreibungen für das Management zu hinterlegen. Es gilt, jede Änderungsanforderung auf Grund ihres Ursprungs auf Berechtigung zu überprüfen, bevor eine Umsetzung in Betracht gezogen wird.

Virtuelle Organisationen können prinzipiell von jedem Nutzer eines Grids erzeugt werden, so es die Richtlinien und Berechtigungen zulassen. In unserem Ansatz haben wir keine Einschränkungen in dieser Richtung vorgenommen. Der Erzeuger einer VO wird automatisch mit administrativen Rechten ausgestattet. Er gilt fortan als der Manager oder Verwalter dieser VO.

Im Rahmen der Verwaltung soll der Lebenszyklus einer VO administriert werden. Dieser besteht aus der Erzeugung, der Initialisierung, dem Betrieb und der Auflösung. Im folgenden werden die Aufgaben, die in den einzelnen Abschnitten auf einen VO-Verwalter warten, aufgezeigt. Diese sollen durch die Implementierung mittels Standard-Tools ermöglicht werden.

Nicht-Existenz. Der Zustand des „Nicht-Sein“ ist der Anfang oder das Ende im Lebenszyklus einer VO. Nur hier besitzt sie keinen Administrator und auch keine sonstigen Eigenschaften. Es sind auch keine Verwaltungsoperationen auf ihr durchführbar.

Entstehung. Ein anderweitig berechtigter Grid-Nutzer beauftragt die Erstellung einer neuen VO. Als Erzeuger wird er dadurch zum Verwaltungsberechtigten. Er erhält die notwendigen Eintragungen und eine Referenz, über die er auf die VO zugreifen kann.

Betrieb. Zur Lebenszeit einer VO sind desöfteren Veränderungen an ihrer Konfiguration vorzunehmen. Hierzu zählen Verwaltungstätigkeiten, die die oben genannten Eigenschaften betreffen. Die Hauptaufgabe ist die Aufnahme neuer Personen und deren Berechtigung. Weiterhin müssen natürlich Ressourcen aufgenommen werden.

Anpassung. Änderungen an den bereitgestellten Diensten, wie deren Auflösung, stellt eine Anpassung im Lebenszyklus der VO dar. Nach einem so grundlegenden Eingriff in die Struktur der VO geht diese wieder in die Betriebsphase über, nun aber mit angepassten Grundeigenschaften.

Auflösung. Die technische Auflösung einer VO geschieht durch einen einfachen Befehl. Eine Zerstörung der in der VO referenzierten Ressourcen muss nicht, kann aber vorgenommen werden.

Bei der *Erstellung* einer VO müssen die Grundinformationen, wie initiale Nutzer und deren Berechtigungen und enthaltene Ressourcen festgelegt werden. Um das zentrale Management zu entlasten, können organisatorische Aufgaben auch an andere Mitglieder weitergegeben werden. Die elektronische Rechtevergabe muss auch hier wieder an eine Prozeß-Richtlinie gebunden sein, um unautorisierter Weitergabe von Rechten vorzubeugen. Noch zu betonen ist, dass in dieser Arbeit nur die Formation einer VO in Betracht gezogen wird. Der interessierte Leser sei für andere Bereiche auf die angegebenen Quellen [Schi 07] verwiesen.

Die Beschränkung auf die WSRF-Interfaces reduziert die Möglichkeit der systemseitigen oder programmatischen Sicherheitsmechanismen drastisch. Auf diese Probleme wird am Ende des Implementierungs-Teils eingegangen.

Beim *Zerstören* ist auch zu beachten, ob die enthaltenen Ressourcen eventuell ebenfalls aufgelöst werden müssen. Dabei ist es relevant, ob in diesen noch eventuell wichtige Zwischenergebnisse gesichert werden müssen. Dies ist wiederum ein verfahrenstechnisches Problem, da auch definiert werden könnte, dass Ressourcen unabhängig von der logischen Struktur einer Virtuellen Organisation weiter existieren können. Dann könnten die Daten in einer Ressource auch für andere Organisationen genutzt werden, in denen die Ressource vielleicht auch enthalten ist, oder in die sie dann aufgenommen wird.

Bei der *Verwaltung von Mitgliedern* und deren Rechte müssen mehreren Anforderungen berücksichtigt werden: es müssen die Voraussetzungen zur Aufnahme zu einer VO (z. B. erfolgreiche Grid-Authentifizierung) vordefiniert werden. Es muss das Recht zur Benutzung der gemeinsamen VO-Plattform den neuen Mitgliedern hinzugefügt werden. Nicht alle Mitglieder sollen zu den gleichen Ressourcen gleich autorisiert werden. Außerdem kann ein Mitglied auch mehreren VOs gleichzeitig angehören. Sein Status und seine Rechte kann

er bei dem Verwalter beantragen und im Falle des Ablaufs seines Zertifikates oder Missbrauch automatisch von den entsprechenden Rechten befreit werden.

Das *Update* einer VO besteht darin, dass die VO auf Anpassungen und Veränderungen rechtszeitig reagiert. Es müssen Muster festgelegt werden, wonach die VO nach Veränderungsnotwendigkeiten in bestimmten Zeitabständen geprüft wird. Für das von gemeinsam von den VO-Mitgliedern festgelegtes Ziel kann die Leistung oder die Ressourcen aus verschiedenen Gründen nicht ausrechend erscheinen. Dafür muss der entsprechende Zuständiger Bestellungen auslösen. Bei der Aufnahme von Personen oder Ressourcen in einer VO muss die Rechtevergabe angepasst werden [SZM 06a].

Noch zu betonen ist, dass in dieser Arbeit nur die Formation einer VO in Betracht gezogen wird. Der interessierte Leser sei für andere Bereiche auf die angegebenen Quellen [Schi 07] verwiesen.

3 Anforderungen

Nach dieser begrifflichen Einordnung sollen nun die Anforderungen an eine Realisierung von Virtuellen Organisationen als WS-Resourcen formuliert werden. Im Folgenden definieren wir ein Szenario, aus welchem wir die wichtige Anforderungen ableiten werden.

3.1 Szenario

Der Wandel des Klimas hat sich in letzter Zeit stark beschleunigt, so dass mittlerweile viele dauerhaften Einflussfaktoren Bestandteile wissenschaftlicher Forschung und Berechnung sind. Schwieriger zu berechnenwird es allerdings bei azyklisch oder einmalig auftretenden Einflussgrößen, die eventuell nicht exakt quantifiziert oder erklärbar sind. Das hat zu Folge, dass sich Berechnungsmodelle aus einer steigenden Vielzahl von Rechengrößen und einer stark wachsenden Menge an Eingabedaten zusammensetzen.

Da der Klimawandel ein globales Problem ist, haben sich mehrere Forscher bereitgestellt, die Klimaveränderung zu analysieren, die Einflußgrössen und deren Korrelationen zu der restlichen Umgebung festzustellen und, falls möglich, zu quantifizieren. Die Faktoren, die das Klima beeinflussen, sind unterschiedlicher Art: wie atmosphärische Schwebstoffe, so genannte Aerosole, Störungen der Meeresströmungen, der Mondeinfluss auf die Gezeiten und damit die großen Meeresströmungen etc.

Zu diesem Projekt werden sich viele Forscher – Physiker, Chemiker, Archäologen, Meteorologen, Klimaexperte, Mathematiker und andere – aller Welt aus vielen unterschiedlichen Organisationen beteiligen. Des Weiteren werden viele verfügbare Hochleistungsrechner in Form von Grid-Netzen zu einem Rechenverbund zusammengeschlossen, um die Kommunikation und Zusammenarbeit zwischen den Teilnehmern, sowie die gemeinschaftliche Datenauswertung zu bewerkstelligen.

Die Einflußfaktoren sollen organisatorisch nach den Forschungsgebiet zugeteilt werden. Außerdem werden die Forscher nach deren Schwerpunkten eingeteilt und zu einem bestimmten Teilprojekt zugeordnet. Das Projekt ist zeitlich beschränkt und die Forscher, die bei dem Projekt beitragen wollen, können jeder Zeit beitreten oder auch das Projekt verlassen.

Bei der Klimaforschung kommt es besonders auf die Berechnung lokaler Veränderungen und Phänomenanalyse an, wobei die Resultate zu einem großen Ganzen zusammengesetzt werden. Die Verwaltung von VOs muss also auf der einen Seite unabhängig von geographischen und politischen Grenzen zentralisiert möglich sein, andererseits auch das verteilte und lokal konzentrierte Management ermöglichen.

3.2 Anwendung auf das VO-Modell

Für den, im Szenario oben beschriebenen, typischen Anwendungsfall bietet die Verwendung von Virtuellen Organisationen eine ideale Möglichkeit der Verwaltung und Strukturierung.

Die Ausgangsbasis sind die real existierende Organisationen, also Firmen, Universitäten etc. Diese Organisationen beschäftigen jeweils einen Mitarbeiterstab, der in die jeweilige hierarchische Struktur eingebettet ist. Des Weiteren verfügt ein Teil dieser Organisationen über Hochleistungsrechner oder anderweitige EDV-Ressourcen, die für Projekte bereit stehen. Dies entspricht der Ebene der Realen Organisationen im VO Management-Modell (siehe Abbildung 1.1).

Um nun die einzelnen, lokal verteilten und separat verwalteten Ressourcen gemeinsam Nutzen zu können, werden diese zu einem globalen Grid zusammengeschlossen. In dieser Stufe geschieht die Verwaltung noch

nicht zwingend zentral, da die Ressourcen weiterhin unter der Obhut ihrer Eigentümer liegen, die sie nur zur Benutzung im Grid freigegeben haben.

Zur Realisierung eines Projektes, wie das oben genannte Klimaforschungsprojekt, könnte nun natürlich das Grid exklusiv verwendet werden. Da es aber keinen Sinn macht, ziemlich viele und rechenstarke, weltweit verbundene Ressourcen exklusiv für ein einzelnes Projekt zu verwenden, werden die sinnvollen und notwendigen Teile in einer Virtuellen Organisation zusammengefasst.

Zunächst müssen dazu die Personen, die in der VO zusammenarbeiten sollen, sowie die Dienste und Ressourcen, derer sie sich bedienen, identifiziert werden. Die Abbildung der realen Ressourcen zu virtuellen, die Teil der VO sind, ist dann Teil der Initialisierungsphase. Hierbei werden auch die Personen in die jeweilige VO aufgenommen.

Während der Initialisierung muss auch die Rolle des Administrators bzw. Managers vergeben werden. Geht man davon aus, dass die integrierten Dienste und Ressourcen bereits über ein standardisiertes Interface verwaltbar sind, liegt es nahe, die VO selbst auch in der gleichen Weise verwalten zu wollen.

Im normalen Betrieb haben selbst innerhalb einer VO nicht alle Personen die gleichen Rechte. Entsprechend ihrer Rolle dürfen sie Verwaltungstätigkeiten ausüben, oder Zugriff auf bestimmte Ressourcen haben.

Das Web Services Resource Framework beschreibt eine einheitliche Methode, Ressourcen und Dienste zu adressieren [GHR 06], auch der Zugriff und die Struktur von Informationen innerhalb von Ressourcen ist festgelegt [GrTr 06]: daher die logische Konsequenz – die Virtuelle Organisation im Rahmen des WSRF als WS-Resource zu implementieren.

Diese Umsetzung ermöglicht es uns, die neu erzeugten VOs mit den gleichen Mitteln wie die einzelnen Ressourcen, als *managed objects*, zu initialisieren, zu betreiben, sie anzupassen und wieder aufzulösen.

3.3 Nichtfunktionale Anforderungen

Im Verlauf dieses Kapitels haben wir ein praktisches Beispiel für die Verwendung von VOs gezeigt. Dieses Beispiel haben wir auf das theoretische Modell von VOs abgebildet. Im Weiteren wollen wir nun nichtfunktionale Anforderungen, die wir für die Arbeit definiert haben, besprechen.

3.3.1 Umsetzung im Rahmen des WSRF

Wie die zu Beginn dieser Arbeit schon verdeutlichte Aufgabenstellung besagt, soll die Implementierung dem Rahmen des WSRF folgen. Dadurch erhalten wir eine definierte Struktur und Schnittstelle, über die eine Verwaltung von VOs mit bestehenden Mitteln möglich ist.

3.3.2 Identifikatoren

Wie beschrieben möchten wir die Zuordnung von Personen zu Rollen oder Gruppen umsetzen. Dabei ist es notwendig, eine Person oder eine Gruppe (durch Identifikatoren) eindeutig referenzieren zu können. Der Identifikator kann dann dazu verwendet werden, um die Mitgliedschaft in einer Gruppe auszudrücken.

Auch für Ressourcen und Dienste, die in einer VO enthalten sind, muss je ein eindeutiger Identifikator existieren. Dafür ist der WS-Addressing-Teil des WSRF [GHR 06] zu verwenden.

3.3.3 Mitgliedschaft

Personen können durchaus Mitglieder einer VO sein, so wie es im Modell der Fall ist, wenn die Personen auf Rollen abgebildet werden, die Teile einer Virtuellen Organisation sind. Tatsächlich aber definieren wir die Mitgliedschaft einerseits über die Abbildung auf eine Rolle (bzw. die der Rolle entsprechenden Gruppe

in einer VO), oder aber über das direkte Zugriffsrecht einer Person auf eine Ressource. Ist einer von beiden Faktoren für eine gegebene VO erfüllt, so ist die Person Mitglied in einer VO.

Daraus läßt sich ableiten, dass es die Möglichkeit geben soll, Personen direkt oder über die Zugehörigkeit zu einer Rolle zum Zugriff auf Ressourcen zu berechtigen.

3.3.4 Überlappung

Durch die Abbildung der Realität, aus den realen Grenzen und Gruppierungen hinaus, auf eine neue Ebene haben Virtuelle Organisationen die Möglichkeit, neue Strukturen und Zusammenschlüsse unabhängig von tatsächlichen Abhängigkeiten zu bilden.

Dabei ist es nicht vorgeschrieben, dass jede reale Person oder Ressource nur auf einen virtuellen Gegenpart abgebildet wird. Es soll also möglich werden, dass sich die Grenzen von zwei VOs überlappen. Dabei kann es unterschiedliche Definitionen für die Überlappung – an Personen oder an Ressourcen – geben.

Eine Überlappung anhand von Personen findet statt, wenn die gleiche Person in mehreren VOs Mitglied ist. In jeder dieser VOs kann die Person andere Rollen einnehmen, und dadurch unterschiedliche Berechtigungen erhalten.

VOs können sich anhand von Ressourcen ausschließlich komplett überlappen. Bei den Sub-VOs besteht die Möglichkeit, auf Ressourcen der übergeordneten VO zuzugreifen. Es ist nicht untersagt, einen Kreis zu schließen, und die Sub-VO gleichzeitig zur übergeordneten VO der ihr übergeordneten VO zu machen. Dadurch können beide VOs gegenseitig auf ihre Ressourcen zugreifen.

3.3.5 Resource Properties

Die Implementation einer VO als WS-Resource beinhaltet die Beschreibung ihrer Eigenschaften als Resource Properties. Bei der Anwendung des Szenarios auf das allgemeine Modell 1.1 für Virtuelle Organisationen, wie oben in 3.2 beschrieben, haben wir die notwendigen Eigenschaften einer VO erkannt. Diese sind im Folgenden beschrieben und stellen daher unsere Anforderung an die Resource Properties einer VO als WS-Resource dar.

Webservices und Ressourcen. Die Basisfunktion eines Grids ist der Zusammenschluss von Ressourcen zu einem größeren Verbund. Für eine Virtuelle Organisation werden dann diese reale Ressourcen auf virtuelle abgebildet. Analog wird mit Diensten verfahren. Wir benötigen also eine Möglichkeit, diese virtualisierten Dienste und Ressourcen in unserer VO zu speichern.

Das WSRF 2.2, in dessen Rahmen wir unsere Implementation durchführen, beschreibt eine WS-Resource als zustandsbehafteten (in Kombination mit einer Ressource) oder zustandsloser Web Service. Dadurch haben wir eine einheitliche Sprechweise für Dienste und Ressourcen, die immer nur über einen Dienst angesprochen werden können.
Weiterhin fordern wir hier Exklusivität – eine WS-Resource kann immer nur einer VO zugehören. Dies ist notwendig, um den Verwaltern der jeweiligen VO die Kontrolle über die Zugriffe anzuvertrauen.

Personen. Personen der Realität werden im VO-Modell auf Rollen abgebildet. Um diese Abbildung implementieren zu können, ist es notwendig, die Ausgangsinformation über die Personen elektronisch vorrätig zu haben. Da dies nicht zwingend in der VO erfolgen muss, leiten wir nur die allgemeine Forderung nach der Existenz einer solchen Informationsquelle ab. Sinnvoll wäre sicherlich, diese Daten außerhalb der VOs zu halten, da eine Person auf mehrere Rollen in mehreren VOs abgebildet werden kann, oder auch zeitweilig ohne eine derartige Mitgliedschaft in einer VO existieren kann.
Dennoch fordern wir, dass jede der abbildbaren Personen zum Grid zugangsberechtigt und gegenüber diesem authentifiziert sein muss. Jede Person besitzt nach der Authentifikation einen eindeutigen Identifikator, durch den sie referenziert werden kann.

Gruppen. Eine Rolle ist die Abbildung einer (möglicherweise leeren) Menge an Personen in die Virtuelle Organisation. Hierbei sind die Rollen für eine VO eindeutig. Es ist aber auch möglich, Rollen ineinander zu verschachteln, um organisatorische Strukturen abzubilden. Dieses Rollen-Modell muss auch in

der Implementierung einer VO als WS-Resource vorhanden sein. Um den Unterschied zwischen Modell und Implementation herauszustellen, werden wir im Folgenden von Gruppen sprechen.
Eine Gruppe soll also eine beliebige Anzahl an Personen enthalten können. Gruppen können auch wiederum Gruppen enthalten. Dies bedingt die Tatsache, dass auch Gruppen eindeutige, referenzierbare Identifikatoren erhalten. Durch den eingeschränkten Scope der Gruppen erhält dieser allerdings nur innerhalb einer VO seine Gültigkeit.

Sub-VOs. Oben haben wir zwei Varianten zur Überlappung von VOs beschrieben. Die Überlappung anhand von Ressourcen geschieht nur durch vollständige Unterordnung einer VO einer anderen. Diese Relation zählt ebenfalls zu den Eigenschaften der organisatorisch höhergestellten VO, da sie dadurch per Definition das Recht auf die eigenen Ressourcen vererbt.

Rechte. Im Rahmen des VO Managements soll es ermöglicht werden, den Zugriff auf einen Web Service bzw. dessen Methoden oder eine Ressource durch Personen oder Gruppen einzuschränken. Dabei gilt ein negativer Ansatz – nur explizit eingetragene Rechte sind verrgeben.
Für die Verwaltung einer VO werden nur in dieser VO eingetragene Zugriffsrechte in Betracht gezogen. Die Verwendung eines beliebig anderen Web Services jedoch wird genehmigt, falls in mindestens einer VO ein Recht auf diesen Web Service und die aufgerufene Methode eingetragen ist.
Ein in einer VO eingetragenes Recht ist nur gültig, falls es sich auf sie selbst oder auf eine der enthaltenen Ressourcen oder Web Services bezieht.

3.3.6 Erhaltung der Dynamik

Wie wir bereits beschrieben haben, ist eine WS-Resource ein Web Service oder ein Web Service in Kombination mit einer Ressource. Ein Web Service alleine ist zustandslos, aber in Verbindung mit einer Ressource kann die WS-Resource einen Zustand speichern.

Die mit dem Web Service verbundene Ressource ist meist die Repräsentation von realer Hardware, deren Zustand über den Web Service manipuliert werden kann. Dabei ist der Web Service meist das reine Interface zur Ressource, auf der der Anwender konkrete Operationen durchführt. Dies können Berechnungen auf einem Großrechner oder Messungen und Steuerungen an Laboreinrichtungen oder Satelliten sein. Dabei stellt also die Ressource ihre Fähigkeiten dem Anwender zur Verfügung.

Eine als WS-Resource implementierte VO entspricht demgegenüber nicht einer konkreten Hardware. Zwar besteht sie aus einem Web Service und einer Ressource, in der die Zustände gespeichert sind. Die Ressource ist allerdings rein virtuell, sie entspricht einem Speicherbereich auf einem Grid Rechner, in dem die Informationen niedergelegt sind.

Durch den Lebenszyklus der Virtuellen Organisation erhält diese WS-Resource eine besondere Dynamik. Während bei einer normalen WS-Resource die Veränderungen in engen Bahnen ablaufen, und auf den Eigenschaften und Fähigkeiten der beinhalteten Ressource beruhen, ist die VO ein rein logisches Konstrukt.

Darüberhinaus hat die Konfiguration der VOs auch Einfluß auf das Gesamtverhalten des Systems. Werden Personen aufgenommen oder entfernt, Rollen definiert oder aufgelöst, oder sogar ganze VOs hinzugefügt oder gelöscht, reagiert das Grid auf die Anfragen vieler Nutzer entsprechend unterschiedlich.

Diese Dynamik ist eine Grundeigenschaft der Virtuellen Organisationen und muss in der Implementierung berücksichtigt und unterstützt werden.

3.3.7 Rahmenbedingungen der Implementierung

Bereits in der Einleitung wurde erwähnt, dass die Umsetzung mit Hilfe des Globus Toolkits 4 erfolgen soll. Diese Anforderung rührt daher nicht aus strategischen Entscheidungen, sondern aus grundsätzlichen Bedingungen.

Prinzipiell gäbe es alternative Umgebungen, wie z.B. das Europäische Unicore[1]. Allerdings ist es nicht Gegenstand dieser Arbeit, ein geeignetes Toolkit zu wählen oder verschiedene zu bewerten. Die Anforderung, die

[1] Weitere Informationen unter http://www.unicore.eu/

VO als WS-Resource mit dem GT4 zu implementieren, wird daher als gegeben hingenommen. Im Allgemein sollen aber die gewonnenen Erkenntnisse natürlich übertragbar sein.

3.4 Funktionale Anforderungen

Die im Abschnitt über VO-Verwaltung dargestellten Tätigkeiten in Bezug auf den Lebenszyklus einer VO manipulieren direkt oder indirekt die Eigenschaften, die wir im letzten Abschnitt als Resource Properties definiert haben. Der Rahmen des WSRF gibt uns den funktionalen Part der Anforderungen vor. Im folgenden Abschnitt werden die Mindestanforderungen an zu implementierende Teile des WSRF aufgelistet und kurz beschrieben.

3.4.1 Erzeugung

Zu Beginn muss eine VO natürlich erzeugt werden können, wofür eine Methode bereitstehen muss. Allerdings ist hierfür keine Vorgabe aus dem WSRF zu entnehmen, die Erzeugung einer VO bleibt also proprietär und ist somit von der weiteren Verwaltung der VO zu trennen. Für diese logische Trennung, die im folgenden Kapitel 5 genauer erklärt wird, verwenden wir die Fabrik-Methode.

3.4.2 Auslesen und Manipulation der Eigenschaften

Um auf die gespeicherten Eigenschaften der VO zugreifen zu können, benötigen wir mindestens `get()`- und `set()`-Methoden. Für den lesenden Zugriff auf die Informationen bietet das WSRF eine Accessor-Methode, die vom Standard sogar für jede WS-Resource zwingend angefordert wird.

Ein VO-Verwalter aber muss ab der Initialisierungsphase auch Änderungen an der VO-Konfiguration vornehmen. Im vorigen Abschnitt haben wir beschrieben, welche Resource Properties unsere als WS-Resource implementierte VO haben soll. Mittels einer WSRF-konformen Methode sollen nun diese Resource Properties auch angepasst bzw. auf einen neuen Wert gesetzt werden können.

3.4.3 Auflösen der VO

Erreicht eine VO das Ziel, sollte sie in der Regel aufgelöst werden. Zur Auflösung einer VO bietet WSRF mehrere Möglichkeiten. Eine davon ist die `ImmediateResourceDestruction`, bei der eine Auflösungsnachricht an die Ressource geschickt wird.

Falls der Zeitrahmen für das Bestehen einer VO gleich vom Anfang an bekannt ist, so kann die Auflösungszeit in die VO eingetragen werden und anschließend automatisch zu dem angegebenen Zeitpunkt aufgelöst werden, ohne dass sich der Verwalter extra darum kümmern muss.

Zur Information der Mitglieder und des VO Verwalters soll eine Methode implementiert werden, die Auskunft über die gesetzte Lebenszeit der VO gibt.

3.4.4 Zugriffskontrolle

Oben wurde bereits das Konzept der Rollen und die daraus folgende Unterscheidung zwischen den Zugriffsrechten erläutert. Dabei werden die Rechte für einzelne Rollen als Eigenschaft der VO gespeichert.

Bei jedem Zugriff eines Clients wird die entsprechende Zugriffskontrolle durchgeführt, d.h. es wird geprüft, ob dieser das entsprechende Recht besitzt. Ist dies nicht der Fall, so darf die Operation nicht ausgeführt werden.

3.4.5 Fehlerbehandlung

Bei den verschiedenen Verwaltungstätigkeiten des VO-Managers können natürlich Fehler in der Implementierung, aber auch Fehler durch bewußte oder unbewußte Fehlbedienung auftreten. Dabei gilt es, diese Fehler zu erkennen und den Manager darüber zu informieren. Optimalerweise gibt die Implementierung einen Verbesserungsvorschlag, um diese Fehler in der Zukunft zu verhindern.

3.5 Lösungsidee

Wie eingangs erwähnt, bieten Virtuelle Organisationen eine Strukturierungsmöglichkeit. Darüber hinaus soll es unsere Lösung ermöglichen, die so gewonnene Struktur auf einfache Weise mit bestehenden Mitteln zu verwalten.

VOs dienen einem konkreten, zeitlich begrenzten Zweck. Für das vorliegende Projekt-Szenario wird eine VO angelegt, in der alle teilnehmenden Forscher und Studenten zusammengefasst sind. Die Implementation bietet hierbei die Möglichkeit, persönliche Daten wie Kontaktinformationen in der jeweiligen VO zu speichern. Ein externer Verzeichnisdienst wäre hierfür allerdings vorzuziehen.

Um einen Identifikator für einen Nutzer zu erhalten, erzwingen wir die Authentifikation gegenüber dem Grid. Danach können wir für den Benutzer aus dem Nachrichtenkontext eine eindeutige ID erhalten, die kryptographisch geprüft wurde.

Gruppierungsmöglichkeiten werden gegeben, um bestimmte Personenkreise nach geographischen, fachlichen oder organisatorischen Gesichtspunkten zusammenzufassen. Eine Gruppe erhält dabei einen Namen, der sie VO-weit eindeutig identifiziert. Es wird auch möglich sein, Untergruppen zusammenzufassen.

Die von Universitäten und Unternehmen beigetragenen Ressourcen sind zu einem oder mehreren Grids verbunden. Diese Kapazitäten können nun ebenfalls der VO hinzugefügt werden. Dadurch erreicht man eine Informationssammlung an einem zentralen Punkt mit einheitlicher Struktur. Bei Ressourcen stützen wir uns auf die vom WSRF vorgegebenen WS-Resourcen, um eine Kombination von Web Service und (optional) Ressourcen speichern zu können.

Auch die Teilprojekte können über Virtuelle Organisationen verwaltet werden. Für jede spezialisierte Forschungsgruppe, die mit Hilfe von gemeinsamen Ressourcen Forschungen durchführt, wird eine VO angelegt. Diese VO wird dann als Teil der gesamten VO deklariert. Dies implementiert die Überlappung anhand von Ressourcen, da die Sub-VO auf die Ressourcen der übergeordneten VO zugreifen darf.

Die hier vorgestellte Lösung bietet eine Möglichkeit zur Erzeugung und Verwaltung der benötigten Organisationseinheiten. Ziel ist es in dieser Arbeit, die Virtuellen Organisationen mit den gleichen Mitteln verwalten zu können, wie die klassische WS-Ressourcen. Daher implementieren wir die VO als WS-Resource gemäß dem Web Services Resource Framework (WSRF).

In der Literatur findet man andere verwandte Projekte, die sich mit einigen Aspekten unseres Themas beschäftigt haben. Die von denen vordefinierten Projekt-Anforderungen, überschneiden sich teilweise mit unseren. Im nächsten Kapitel 4 werden ähnliche Projekte ausführlicher dargestellt.

4 State-of-the-Art

Im Kapitel 3 haben wir einige Anforderung definiert, die wir im Laufe unseres Projektes umsetzen wollen. Bevor wir mit der Erfüllung der gesammelten Anforderungen beginnen, wollen wir untersuchen, ob es dazu bereits Ansätze oder Lösungen gibt, welche Anforderungen bei diesen Projekten gesetzt und erfüllt wurden, und inwieweit sie unse wurdenrschneiden.

Grid Computing ist derzeit ein „heißes Pflaster“in der Forschung. Wie einleitend erwähnt, ist die Formalisierung Virtueller Organisationen selbst noch nicht abgeschlossen. Es ist schwierig zu erkennen, ob andernorts vielleicht dergleiche Ansatz wie bei uns verfolgt wird, da von den einzelnen Projekten erst nach Abschluss der Forschungen konkrete Ergebnisse präsentiert werden.

Im folgenden stellen wir diverse verwandte Projekte vor. Diese werden am Ende dieses Kapitels abschließend zusammengefasst.

4.1 WSRF based Virtual Organization Middleware

Bestes Beispiel für ein paralleles Projekt ist das Projekt Asif Akram, der vor ein paar Monaten eben auch die Umsetzbarkeit Virtueller Organisationen im WSRF untersucht hat [AkAl 07].

Akram hat keine Implementierung durchgeführt, sondern nur die Sinnhaftigkeit einer Umsetzung untersucht. Er konzentriert sich dabei auf die Implementation als WS-ServiceGroup [MSB 06] und das Web Services Distributed Management (WSDM) [BMW 06]. Letzteres wurde von OASIS als Standard verabschiedet und es spezifiziert eine Management-Struktur für WSRF-konforme Ressourcen und Web Services.

Neben der Vermischung von abstrakter Sichtweise und Implementierungsdetails werden allerdings auch keine genauen Managementanforderungen gestellt, die letztendlich erfüllt werden sollen. Allgemein lässt sich sagen, dass ein paar Ideen ähnlich den in dieser Arbeit betrachtet wurden, allerdings bei weitem nicht so ausführlich.

Da keine weiteren Arbeiten gefunden wurden, bei denen sich das Thema so direkt überlappt, werden im folgenden einige verwandte und themennahe Projekte vorgestellt. Einige davon könnten als Ergänzung oder Erweiterung unserer Spezifikation und Umsetzung dienen.

4.2 PERMIS

Authentifizierung der Kommunikationspartner mit starker, standardisierter Kryptographie, wie mit X.509 Zertifikaten, ist der erste notwendige Schritt für sichere Kommunikation. Darauf folgt aber unmittelbar die Frage, welche Operationen die andere Kommunikationsseite ausführen darf. Die X.509-Spezifikation sieht in ihrer vierten Ausgabe dazu eine Privilege Management Infrastructure (PMI) [ChOt 02] vor. Die PMI ist dabei das äquivalent zu einer Public Key Infrastructure (PKI) [Open 00] bei der Authentifizierung. Entsprechend einem öffentlichen Schlüssel, der auf kryptographischem Weg die Identität nachweist, werden hier Attribut-Zertifikate (Attribute Certificate (AC)) verwendet. Diese drücken die Bindung zwischen einer Person, oder allgemein einer Kommunikationseinheit, und ihren Berechtigungen aus.

Privilege and Role Management Infrastructure Standards Validation (PERMIS) [ChOt 03] wurde als Projekt ins Leben gerufen, um bereits bestehende PKI-Infrastrukturen um die PMI Komponente zu erweitern. Somit wird eine komplette Authentifizierungs- und Autorisierungskette geschafft [Chad 01]. Dabei wird ein rollenbasiertes Berechtigungsmodell (Role Based Access Control (RBAC)) verwendet. Einer Person wird vom Administrator eine oder mehrere Rollen zugewiesen.

Um die Zuweisung von Rollen und die Weitergabe über Vertrauensstellungen und Signaturen zu ermöglichen, werden X.509-Zertifikate, also die oben genannten ACs, für Berechtigungen verwendet. Ein Nutzer hat dabei die ACs, die seine Rollenzugehörigkeit bestätigen. Die Rolle besitzt ebenfalls ACs, in denen die Berechtigungen für die Rolle niedergeschrieben sind. Eine hierarchische Struktur von Rollen ermöglicht es, Vererbungen von Rechten über die Rollen vorzunehmen und gleichzeitig die gespeicherte Datenmenge zu begrenzen.

Die PERMIS wurde von Information Systems Security Research Group (ISSRG), an der University of Kent (UK) [1] implementiert. Das Tool ist eine in Java geschriebene Open-Source Autorisierungssoftware, die auf in XML-beschriebenen Policies basiert. Der System-Administrator darf Policies erzeugen, in denen die Rollen und Zugriffsrechte beschrieben werden. Der VO-Administrator kann seinerseits weitere Policies erzeugen, die die Zuordnung von Rollen zu den Benutzern beinhalten.

Das Zertifizierungskonzept ist sehr nachteilig, weil die Zertifikate bei dem Eingang einer Autorisierungsanfrage nicht dynamisch erzeugt werden. Sie liegen nämlich stattdessen bereits für jeden Benutzer für die Autorisierung bei einer bestimmten VO vor. Hat sich der Benutzer authentifiziert, wird das Zertifikat aus der Repository, eine zentralle Sammlung von Zertifikaten, geholt.

Jedes Mitglied darf nur ein einziges Zertifikat anfordern, das die Mitgliedschaft einer VO erweist. Wird der Zutritt einer anderen VO erforderlich, muss sich der Benutzer daher erneut authentifizieren.

Auch betrachtet PERMIS nur die Berechtigungsfragen beim Zugriff auf virtuelle Ressourcen. Die Formation und restliche Aspekte des VO Managements bleiben unberührt.

4.3 VOMS

Der VO Membership Service (VOMS) [ACC+ 04] wird ausschließlich nur zur Strukturierung von Personen und Organisationen einer Virtuellen Organisation verwendet, und zwar, um die Informationen über den Status eines Benutzers aufzurufen. Dabei werden alle Mitglieder-Properties zu einem Attribut-Zertifikat hinzugefügt. Dieses Zertifikat beinhaltet keine Schlüssel im Vergleich zum x.509-Zertifikat. Es wird über die VOMS-Serverseite mit dem privaten Schlüssel seines X.509-Host-Zertifikates signiert.

Ein Proxy-Zertifikat kann mehrere Attribut-Zertifikate von verschiedenen Virtuellen Organisationen enthalten. Somit kann der Benutzer sich dann an verschiedenen VOs anmelden und mehrere VO-Attribute in einem Proxy-Zertifikat nutzen. Eine Integration mit einer Globus Grid Version zur Verwaltung von VOs ist zwar möglich, aber bis jetzt noch kein Integrationsprojekt bekannt.

Für jede VO wird ein VOMS-Server eingerichtet, der dann alle Benutzerkonten und deren Rechte beinhaltet. Um alle diese Daten zu speichern und zu verwalten, benötigt der VOMS-Server eine relationale Datenbank, wo neben den aktuellen auch vergangenen Zuständen abgelegt werden. Zur Registrierung und Administration von VOs können Web Services verwendet werden. Die Installation vom VOMS ist leider sehr mühsam. Im Vergleich zum einfachen Einrichten einer Datenbank oder eines Webservers, sind die weiteren systembezogenen Anforderungen sehr schwer zu erfüllen. Viele angeforderte Systempakete sind in unterschiedlicher Versionen verfügbar und sie müssen mit dem System abgestimmt werden.

Ein VOMS-Defizit besteht darin, dass die VOMS-Server die Mitgliederinformationen untereinander nicht tauschen können. Eine Art Vererbung für das Einrichten und Verwalten, und Export- und Importfunktion für das Wechsel der Mitgliedschaft in eine anderen VO, wäre für die Zukunft wünschenswert. Die Aufnahme eines Mitglieds in eine VO ist mit sehr vielen administrativen Aufwand verbunden. Dieses Problem wird in der VOMS-Erweiterung (VOMRS) teilweise gelöst.

Ein anderes Problem beim VOMS-Ansatz ist die Tatsache, dass keine Grid-Ressourcen 2.5 durch diese Software verwaltet werden, also keine Informationen über die Ressource aufgerufen werden können. In unserem Projekt werden die Informationen über die Ressource als Eigenschaften gespeichert, die mit Hilfe von WSRF-Operationen [GrTr 06] aufgerufen werden können.

[1] Weitere Informationen unter http://sec.cs.kent.ac.uk/

4.4 VOMRS

Der VOM Registration Service (VOMRS) [VOX 04] bietet die Möglichkeit, Nutzer einer Virtuellen Organisation aufzunehmen und die Informationen und Koordination dieses Vorgangs zwischen den Verwaltern zu ermöglichen.

VOMRS ermöglicht also die Tasache, beliebige Informationen zu jedem Mitglied zu speichern. Falls eine Person für eine bestimmte Zeit kein VO Zutrittsrecht besitzt, wird der VOMS-Administrator den Teilnehmer komplett entlassen, um später wieder zu registrieren. VOMRS kann den Mitgliedern zeitlich die Rechte entziehen, um dann zu einem späteren Zeitpunkt dieselbe Rechte freizugeben.

VOMRS bietet einen einfachen Workflow von der Registrierung, Bestätigung bis hin zur Verwaltung der Nutzer. Kernpunkt ist dabei ein zentraler Server, auf dem ein Serverprozess pro Virtueller Organisation läuft.

Die Virtuelle Organisation ist hier sehr statisch, das einzige dynamische Element ist der Nutzer. Wir fordern aber eine komplette Dynamik, beispielsweise sollten auch die Ressourcen dynamisch bleiben, denn sie sollten jederzeit zu einer VO aufgenommen bzw. aus einer VO entlassen werden. Auch ist die Zentralisierung der Verwaltung eher gegensätzlich zur globalen Existenz Virtueller Organisationen.

4.5 Liberty

Betrachtet man im VO-Modell die „reale“ Ebene als Ausgang für die Abbildungen auf die „virtuelle“(siehe Grafik 1.1), fällt auch hier eine Unterteilung auf. In der Realität sind Personen und Ressourcen fest einer Organisation oder Einrichtung zugeordnet. Die Abbildungen auf eine virtuelle Welt entsprechen nur Definitionen.

Will sich eine Person im Grid über Konzerngrenzen hinweg „bewegen“, benötigt sie Mechanismen zur Authentisierung. Auf der Basis einer Vertrauensstellung entsteht nun eine Kette, über die sich eine Person an einer weit entfernten Ressource im Grid anmelden. Die Informationen über die Identität und Attribute werden mittels der Security Assertion Markup Language (SAML) [KMC+ 05] in SOAP-Nachrichten ausgetauscht.

Für die Bildung von Virtuellen Organisationen ist dieses Problem in besonderem Maße vorhanden, da durch die Abbildung auf die virtuelle Ebene die realen Grenzen aufgehoben werden müssen. Liberty ist eine von Liberty Alliance entwickelte Spezifikation [Kemp 04]. Sie definiert Mechanismen, Nachrichten und Verfahren zur föderierten Authentifizierung.

Dabei wird die Autorisierung, im Vergleich zu unsere Umsetzung , überhaupt nicht betrachtet. Es wird in der Spezifikation nicht vorgegeben, wie das Login-Interface aussehen, oder auch in welcher Sprache die Spezifikations-Umsetzung durchgeführt sein sollte. Die Spezifikation von Liberty steht jedem frei zur Verfügung, die Implementierungen sind aber mit Kosten verbunden. Deswegen werden Liberty-Produkte eher im kommerziellen Bereich und staatlichen Institutionen eingesetzt.

Voraussetzung für den Liberty-Einsatz ist, dass jeder Benutzer bei den einzelnen Partnern über ein Benutzerkonto verfügt, wobei gegenüber anderen Partnern die Authentisierung über den Simplified Single Sign-on erfolgen würde. Alle Partner tauschen untereinander die Autorisierungsinformationen mit Hilfe von Web Services Framework. Die Zugriffsrechte des Benutzers auf die Information werden aber vom lokalen Access-Manager vergeben.

Wir haben für unseren Ansatz gefordert, dass die Nutzer bereits gegenüber dem Grid bekannt und authentifiziert sind. Die Überlegungen der Liberty Alliance kommen aber natürlich dem globalen Gedanken der Virtuellen Organisationen entgegen. Liberty könnte die Grundlage bieten, auf der unsere VO Implementation aufbaut. Eine Alternative zu Liberty betrachten wir im nächsten Abschnitt.

4.6 Shibboleth

Auch Shibboleth [Shib 01] strebt, wie das oben beschriebene Liberty, eine verteilte Identifizierung an. Während aber Liberty dabei die genaue Identität der Person herauszufinden versucht, begnügt sich Shibboleth mit dem

Wissen, dass die Identität von einer vertrauenswürdigen Institution geprüft wurde.

Shibboleth verwendet dazu ebenfalls SAML [KMC+ 05] und SOAP [MiLa 07]. Es ist eine Open-Source-Implementierung eines verteilten Systems zur inter-institutionellen Nutzung von zugangsgeschützten (Autorisierung und Single-Sign-On Authentisierung) Web-Ressourcen. Das Projekt wird vom Middleware Architecture Committee for Education (MACE) [2] entwickelt.

Ein Shibboleth Kommunikations-Aufbau erfolgt zwischen dem Identity Provider (IdP) und dem Service Provider (SP). Der IdP basiert auf dem Identity Management (IdM) der Heimat-Einrichtung des Nutzers, der SP steht für die Web-Ressource. IdPs und SPs bilden üblicherweise auf nationaler Ebene eine *Föderation*, deren Policy die Vertrauensbasis der teilnehmenden Partner ist. Bei dem Zugriff auf eine Ressource wird geprüft, ob der entsprechende Interessent schon authentifiziert ist. Wenn nicht, wird er weiter an einen Lokalisierungsservice, wo er seine Heimat-Einrichtung auswählen kann, weitergeleitet und da zu Passworteingabe aufgefordert. Somit wird die Heimateinrichtung einen digitalen Ausweis, den sie an den Anbieter zur Prüfung weiterschickt, erzeugen. Bei der Autorisierungsanfrage fragt der Anbieter bei der Heimatseinrichtung des Interessenten, ob der Zugriff erlaubt ist.

Wie wir in der Einleitung gehört haben, wird unsere Implementation im Rahmen des Globus Toolkits erfolgen. GridShib [SBB+ 06] ist eine Brücke zwischen Shibboleth und Globus, die in Kooperation zwischen der Globus Alliance und dem Internet2 Konsortium [3] entstanden ist. Da die verteilte Authentifizierung und globale Gültigkeit von Identitäten oder daraus abgebildeten Rollen eine absolute Notwendigkeit für ernst gemeinte Umsetzungen von VOs sind, bietet es sich an, unsere Globus-Implementierung in dieser Richtung, um GridShib zu erweitern.

4.7 CAS

Der *Community Authorization Service (CAS)* [PWK+ 02]-Managementansatz wird zur Ressourcen-Verwaltung verschiedener Virtuellen Organisationen verwendet. Die Software bietet die Möglichkeit, Zugriffe auf bestimmte Ressourcen zu regulieren. CAS wurde im Rahmen des Globus Projekts entwickelt. Die Authentisierung verfolgt über ein Public-Key Verfahren. Es wird auch die Globus Security Infrastructure (GSI) des Globus Toolkits eingesetzt. Wie beim VOMS wird hier nach der Authentisierung ein Proxy-Zertifikat erstellt, das mit zusätzlichen Informationen, nämlich den Benutzerrechten, angereichert wird.

Im Unterschied zum VOMS, wo die Attribute allgemein gehalten werden und eher Rollen verwendet werden, woraus dann globalen Zugriffsrechte folgen, werden Zugriffsrechte hier nur in *low-level* Bereich durch die CAS-Attribute spezifiziert (z.B. Zugriffsrechte auf eine Datei).

Der Einsatz vom CAS in einer Globus-Umgebung ist sehr mühsam. Alle Grid Dienste müssen aufgrund der CAS-Zertifikate, die nicht dem Globus-Standard entsprechen, zunächst modifiziert werden. Der Globus-Standard ermöglicht jedem Dienst, die Information über den Eigentümer des Zertifikates ohne Probleme zu lesen. Bei den CAS-Zertifikaten ist diese Information ganz anders codiert und alle Globus-Dienste müssen deswegen erstmal so angepasst werden, dass die CAS-Zertifikate auch hier verarbeitet werden können.

Ein weiterer Nachteil von CAS ist, dass es nur einzelne Genehmigungen erlaubt, jedoch keine Gruppen oder Rollen ermöglicht. Da erschwert die Arbeit eines Administrators sehr, da es erst über die Einteilung in Gruppen und Zuweisung von Rollen möglich wird, eine vernünftige Hierarchie innerhalb einer Virtuellen Organisation aufzubauen.

Wir haben CAS hier erwähnt, da es verspricht, die Autorisierung von virtuellen Communities leisten zu können. Wie wir gesehen haben, ist der Funktionsumfang unzureichend und in eine falsche Richtung dirigiert. Gerade die verwendeten *low-level* Berechtigungen auf Benutzerbasis vertragen sich nicht mit dem Rollenkonzept der Virtuellen Organisationen.

[2] Weitere Informationen unter http://middleware.internet2.edu/MACE/
[3] Weitere Informationen unter http://www.internet2.edu/

4.8 DyVOSE

DyVOSE [SKSW 06] war ein zweijähriges Projekt, das von Mai 2004 bis April 2006 lief. Finanziert wurde es von verschiedenen, im Joint Information Systems Committee zusammengeschlossenen Universitäten.

Ausgangspunkt für DyVOSE war die Beobachtung von Einschränkungen im Umgang mit Benutzerzertifikaten und einer PKI im Grid-Umfeld. Daher wurden lokale Authentifizierungmethoden für entfernte Anmeldung getestet, die das oben beschriebene Shibboleth als Transportprotokoll nutzen sollten.

Es wurde festgestellt, dass man eine ebenso dezentrale Autorisierungsinfrastruktur benötigt wird. Allerdings war die Skalierbarkeit bei existierenden Systemen nicht sichergestellt. Es sollte daher auf der Basis von PERMIS [ChOt 03] ein Schema der deligierbaren Autorisation implementiert werden. Dies würde helfen, die Dynamik bei der Bildung von Virtuellen Organisationen abzubilden, da die Rollen und dadurch entstehende Berechtigungen weitergegeben werden könnten.

Im Zuge des Projektes wurde PERMIS um eine dynamische, vererbbare Berechtigungsstruktur erweitert. Diese wurde als *Privilege Management Infrastructure* (siehe auch 4.2) bezeichnet.

Das Ziel von DyVOSE wurde erreicht, einige Lücken sind trotzdem zu erwähnen, wie komplizierte Handhabung von Zertifikaten, mangelnde Dokumentation des closed-source Codes etc. [SKSW 06].

Im Rahmen dieses Projektes wurden viele Anforderungen bezüglich Authentisierung und Autorisierung definiert und darausgestellte Problemstellungen im Laufe des Projektes in eine Lösung umgesetzt. Dabei wurden andere Aspekte einer VO vernachlässigt, z. B. die Möglichkeit Ressourcen- oder Personen-Informationenen aufzurufen.

4.9 Zusammenfassung

Wie wir in unserer nicht abschließenden Übersicht in diesem Kapitel gesehen haben, beschäftigen sich einige Projekte mit der Implementierung Virtueller Organisationen. Allerdings wissen wir nur von Asif Akram, dass er sich die Umsetzung im Rahmen des WSRF zumindest überlegt hat. Die anderen Ansätze sind teils proprietär, teils auf anderen Standards basierend. Dabei ist auch jeweils, verglichen mit den von uns gestellten Anforderungen, nur ein Teil implementiert.

Die Forschungsgruppe des D-Grid hat ein Thesenpapier erstellt, das unter sehr detaillierten Gesichtspunkten die Systeme Liberty, Shibboleth/ GridShib und VOMS miteinander vergleicht [Proj 05]. Das zitierte Dokument enthält dabei ausführliche Informationen in tabelarischer Form, mit einer direkter Gegenüberstellung, aus der weitere Informationen zu entnehmen sind.

Der Fokus der hier vorgestellten Projekte liegt auf der Benutzerverwaltung, beispielsweise beschäftigen sich das VOMS- und Permis-Projekt ausschließlich mit Authentifizierung bzw. Autorisierung. Die Bildung einer VO entspricht meist der Berechtigung einer Gruppe von Personen auf bestimmte Ressourcen und Dienste. Dieser sehr statische Ansatz entspricht nicht unseren Anforderungen. Die von uns geforderte Dynamik sowohl von Personen als auch von Ressourcen, sowie die Unabhängigkeit der Verwaltung von den realen Institutionen ist nicht gegeben.

Andererseits sind die Methoden der föderierten Authentifizierung (z. B. bei Liberty und Shibboleth) und später auch Authorisierung durchaus betrachtenswert. Wir haben gefordert, dass ein Nutzer bereits durch das Grid authentifiziert ist, da dies eine meist lokale Angelegenheit ist. Natürlich muss aber der Nutzer über eine Grid-weit gültige Identität verfügen, um vollständig auf seine Rolle in der virtuellen Ebene abgebildet werden zu können. Eine entsprechende Infrastruktur wird also implizit vorausgesetzt.

Weiterhin bieten Methoden der verteilten Berechtigung, wie sie DyVOSE bietet, eine notwendige Struktur, die wir in unsere Umsetzung einbetten könnten. Allerdings ist dies nicht das Hauptaugenmerk dieser Arbeit, wir werden uns im nächsten Kapitel 5 hauptsächlich mit der Umsetzung Virtueller Organisationen als WS-Resourcen beschäftigen. Dabei wird eine mögliche Erweiterung oder Integration mit föderierter Authentifizierung und Authorisierung mit einem der hier vorgestellten Projekte als mögliche Weiterentwicklung im Auge behalten.

5 Entwicklung einer VO als WS-Resource

Wie im vorigen Kapitel gezeigt, werden Virtuelle Organisationen zur Strukturierung und Verwaltung einer Gruppe, bestehend aus Personen und Ressourcen, genutzt. Ein wichtiger Aspekt bei der Implementierung ist dabei, die Virtuelle Organisation als eine WS-Resource 2.2 zu modellieren. Der WS-Resourcen-Ansatz wird vom WSRF vorgeschlagen, da somit die Möglichkeit besteht, Zustände der Ressourcen zu speichern. Die genauere Beschreibung weiterer Anforderungen ist im Kapitel 3 zu finden.

Mehrere Voraussetzungen sind für dieses Projekt zu berücksichtigen. Vorausgesetzt wird eine aufgebaute Grid-Infrastruktur, auf die bei der Erstellung dieser Arbeit zugegriffen wird. Außerdem sollte unser Ansatz im Kontext des Globus Toolkits 4 auf seine Tragfähigkeit untersucht werden. Im Vergleich zu vielen anderen Werkzeugen, wie Uniform Interface to Computing Resources (Unicore) [1], gLite [2] etc., ist das Globus Toolkit heutzutage weltweit sehr verbreitet und hat sich fast wie ein Middleware-Standard durchgesetzt.

In diesem Kapitel wollen wir zunächst das Design der WSRF-konformen VO darstellen. Danach wird die Umsetzung der gewählten und bis hierhin theoretisch fundierten Lösung ausführlich beschrieben. Gleichzeitig wird die Software-seitige Voraussetzung dargestellt, die noch zusätzlich für unsere Implementation benötigt wird.

Der Text enthält im Verlauf dieses Kapitels des öfteren Quellcode Ausschnitte. Diese sind teilweise gekürzt oder in Pseudocode zusammengefasst. Auf der beiliegenden CD-ROM findet der interessierte Leser die komplette Implementation.

5.1 Design der WSRF-konformen VO

Nachdem wir die Umgebungsbedingungen geklärt haben, nähern wir uns nun ein Stück weit der Implementierung. Bevor die eigentliche Programmierarbeit an einzelnen Klassen beginnt, sind verschiedene Design-Aspekte zu entwickeln und zu definieren. Diese wollen wir in diesem Abschnitt erklären.

Das grundlegende Design bzw. Implementierungsziel war die Realisierung von VOs als WS-Resourcen. Daher beschreiben wir im ersten Unterabschnitt, welche Strukturen und Mindestvoraussetzungen dieses Ziel vorschreibt.

Im Weiteren Verlauf erklären wir die Fabrik-Methode, die vom GT4 als Design-Pattern zur Erzeugung von neuen Ressourcen vorgeschlagen wird, aber auch in der Aufgabenstellung gefordert wurde. Die Implementierung zeigt später auch die genaue Umsetzung auf.

Nicht ausführlich beschrieben, aber bereits in den vergangenen Kapiteln entwickelt, wurde unsere Vorstellung der Sicherheitsarchitektur. Daher beschreibt der Abschnitt über Zugriffskontrolle, welche Maßnahmen wir im Überblick einsetzen.

Um die Rechte und auch Mitgliedschaften den Personen zuzuordnen, mussten wir ein System für Identifikatoren auswählen. Diese werden im darauf folgenden Abschnitt behandelt.

Letztendlich erklären wir noch kurz die Technik der Qualified Names (QNames), die im WSRF-Umfeld für eindeutige Bezeichner sorgen.

[1] Weitere Informationen unter http://www.fz-juelich.de/zam/grid/unicore
[2] Weitere Informationen unter http://glite.web.cern.ch/glite/

5.1.1 Struktur-Design

Eine Frage zu Beginn der Implementierungsarbeiten betraf die Modularität der Umsetzung. Es wäre denkbar, die einzelnen Komponenten einer VO, wie Personen und Gruppen, als eigenständige WS-Resourcen zu realisieren. Dies würde einer direkteren Abbildung der Realität in die Virtualität entsprechen. Allerdings wird dabei nicht nur der Implementierungs- sondern auch der Administrationsaufwand bedeutend höher. Jede dieser Ressourcen müsste extra überwacht, die Abhängigkeiten untereinander ständig geprüft werden.

Aus diesem Grund haben wir uns für einen monolithischen Aufbau mit einer WS-Resource für eine VO entschieden. Die Elemente in der VO werden als Eigenschaften in der WS-Resource beschrieben. Somit bildet jede VO eine administrative Zone.

5.1.2 Design als WS-Resource

Ein wichtiger Aspekt bei der Implementierung ist also die Virtuelle Organisation als eine WS-Resource 2.2 zu modellieren. Der WS-Resourcen-Ansatz wird von den WSRF 2.2-Entwicklern vorgeschlagen, da somit die Möglichkeit vorhanden ist, Zustände der Ressourcen zu speichern. Bezüglich der WS-Resourcen stellt das Framework einige Anforderungen. Die Standards spezifizieren also Mindesteigenschaften einer WS-Resource. Im folgenden Listen wir die relevanten Eigenschaften einer WS-Resource bezüglich Syntax und WSRF-Operationen auf und spezifizieren ihre Herkunft, sowie die Tatsache, ob sie zwingend oder nur optional erforderlich sind:

- Die zugehörige Ressource muss über eine eindeutige ID adressierbar sein. Diese ID ist zwingend erforderlich laut Spezifikation [GKM+ 06].
- Die Null oder mehr Eigenschaften der Ressource müssen in einem ResourcePropertyDocument, einer XML Struktur, abgelegt sein [GKM+ 06, GrTr 06]. Die atomaren Eigenschaften sind die Kind-Elemente dieses Dokumentes.
- Eine WS-Resource muss die Nachricht `GetResourceProperty()` unterstützen [GrTr 06]. Optional, allerdings zum aktiven Verwalten notwendig, ist die Nachricht `SetResourceProperty()`. Letztere ist für den schreibenden Zugriff auf Ressourcen notwendig, der Standard schreibt aber nur die Lese-Methode zum Zugriff vor.
- Da wir das Factory-Pattern 5.1.3 zum Erzeugen von VOs verwenden, WSRF-Empfehlung, ist es zwingend eine `create()`-Methode zu implementieren. Eine `destroy()`-Funktion zur Auflösung einer VO ist nicht vorgeschrieben.

Alle andere WSRF-Methoden sind natürlich optional, aber von den WSRF-Entwicklern trotzdem als hilfreich geschätzt und daher empfehlenswert, beispielsweise:

- Die Ressource kann einen Lebenszyklus besitzen. Falls sie einen solchen hat, kann sie Methoden zur geplanten oder sofortigen Auflösung implementieren. Die Planung findet hierbei über die Angabe einer Zeit statt. Diese wird in der Ressource vermerkt und kann auch abgefragt werden. Nach Ablauf der Zeit kann die Ressource aufgelöst werden. Zumindest sollte angenommen werden, dass die Ressource ab dem gegebenen Zeitpunkt nicht mehr existiert, unabhängig von der tatsächlichen Auflösung.
- Es ist möglich und vernünftig, die WS-Resource über eine Endpoint Reference zu adressieren. Eine solche EPR muss eindeutig sein, sich also auf exakt eine Ressource beziehen. Sie besteht aus der URL des Instance Services sowie der ResourceID (ResourceKey) [GHR 06]. Ein Beispiel für eine solche EPR ist in Anhang I gegeben.

Da wir die Virtuelle Organisationen als WS-Resourcen implementieren wollen, ist es notwendig, die vom WSRF vorgeschriebene Syntax und Operationen bezüglich zustandsbehafteter Ressourcen auf unsere Entities abzubilden.

5.1.3 Fabrik-Methode

Ein anderer Aspekt unseres Designs ist vorzudefinieren, wie die Virtuelle Organisationen zu erzeugen sind. Da wir „multiple resourcen“ erzeugen wollen, haben wir die VO Bildung durch die Fabrik-Methode, laut Empfehlungen von WSRF-Entwicklern geleistet. Multiple Resourcen sind Ressourcen, die sehr oft erzeugt werden und jeweils die gleiche Basisstruktur haben sollen [Akra 06].

Ein VO Verwalter adressiert einen Web Service und möchte dort eine Methode aufrufen, die ihm eine Referenz auf eine neue VO zurück gibt. Dabei sollen die Aktivitäten im Hintergrund vor ihm verborgen bleiben, ebenso die Instantiierung konkreter Klassen und deren Registrierung.

Dieses Muster ist im Bezug auf Objekterzeugung fundamental und ermöglicht die definierte Instantiierung von Objekten. Bei diesem Pattern (siehe auch 5.4) werden Objekte nicht direkt, sondern mit Hilfe einer so genannten „Fabrik“ erzeugt. Ein Client stellt eine Anfrage an einen Factory Service (hier VOFactoryService), der dann eine Ressource (hier VOResource) erzeugt.

Der Factory Service erstellt die Ressource in einem serverseitigen Resource Home (hier VOResourceHome). Die von dort erhaltene eindeutige ResourceID wird an den Client, zusammen mit der Adresse des Instance Service, zurückgeschickt. Letzterer bietet alle zur Manipulation des Objektes verfügbaren Methoden an. Der Aufrufer kann jetzt über die angebotenen Schnittstellen unterschiedliche Informationen der VO abfragen oder ändern (siehe auch weitere Abschnitte des Kapitels).

5.1.4 Zugriffskontrolle

Die Mitglieder in einer Virtuellen Organisation besitzen nicht alle die gleichen Rechte auf die enthaltenen Ressourcen. Daher ist es notwendig, für unsere Implementation auch eine Konfigurationsmöglichkeit und Überprüfung der Zugriffsrechte einzurichten.

Das Globus Toolkit 4 sieht hier eine Authorisierungs-Kette vor, bestehend aus Informationssammelstellen, Policy Information Points (PIPs) und Entscheidungsstellen Policy Decision Points (PDPs). Welche dieser Stellen zu Rate gezogen werden, ist für jeden Web Service konfigurierbar. Wir gehen davon aus, dass für alle relevanten Services die von uns entwickelten PIPs und PDPs in die Entscheidungskette eingefügt werden (siehe auch 5.4.4 und 5.4.5).

Weiterhin besitzt die Entscheidungskette nur Informationen über die aktuell aufgerufene WS-Resource. Um die aus der Mitgliedschaft in verschiedenen VOs erwachsenden Berechtigungen zu sammeln, werden diese alle nach Informationen über den Aufrufer durchsucht. Dies sollte durch eine performantere Lösung ersetzt werden, beispielsweise durch die Erweiterung der hier implementierten *VOResource* als ServiceGroup [MSB 06] Für eine geringe Anzahl an Virtuellen Organisationen ist der Mehraufwand des Durchsuchens allerdings kaum zu bemerken.

5.1.5 Identifikatoren

Für den Menschen sind Namen viel einprägsamer als Nummern oder kryptische Ziffern und Buchstaben-Kombinationen. Eine Gruppe, Universität oder ein Projekt werden immer mit einem bestimmten Titel erwähnt. Daher verwenden wir an der Schnittstelle zum Anwender als Identifikatoren für Gruppen und die Virtuellen Organisation deren Namen. Dieser ist somit eindeutig, eine Verwechslung gleich benannter Strukturen ist ausgeschlossen.

Eine Person hat in der Welt des GT4 aus ihren Zertifikaten erwachsende Identifikatoren, die so genannten „Principals“. Diese verwenden wir, um die Mitgliedschaft in Gruppen, Zugriffsberechtigung und die Verknüpfung mit den Details der Person herzustellen.

Über sie wäre es möglich und auch sinnvoll, die eigentliche Verwaltung der Personen und deren Eigenschaften in ein externes, vielleicht schon bestehendes Verzeichnis auszulagern. Bei unserer Implementation haben wir beispielhaft eine Menge an Informationen über Personen direkt in einer VO gespeichert. Sofort erkennbar ist die Dezentralisierung der Information, die in einem Punkt konzentriert sicherlich besser aufgehoben wäre.

Dies ist eine weitere Verbesserungsmöglichkeit unseres Ansatzes, der allerdings die geforderte Funktionalität voll erfüllt.

5.1.6 Client

Weiterhin müssen wir auch den Client und seinen Zugriff auf die Mittel designen. Ein Client greift in unserem Fall mit Hilfe von WSRF-Operationen auf die Ressourcen zu. Die Funktionalität der Operationen wird durch den *Providern* 5.4.2, die vom GT4 zur Verfügung seit kurzem gestellt sind, durchgeführt.

Die Implementierung des Clients stellt eine hierarchischen Menü-Struktur dar, die den Baumknoten des Resource Property Documentes (RPD) entspricht. Ein RPD existiert im Allgemeinen nicht, dessen Struktur wird aber in der XML-Beschreibung des Instance Services 5.3.2 beschrieben.

5.1.7 Fehlerbehandlung

Java [JBC+ 06] ist eine stark auf Ausnahmefehler (engl. Exceptions) ausgerichtete Sprache. Schon bei der Kompilierung wird geprüft, ob vorgegebene Ausnahmen einer aufgerufenen Methode auch abgefangen werden. Natürlich gibt es auch nicht zwingend zu behandelnde Fehler, diese werden dann solange in der Aufruf-Hierarchie zurückgegeben, bis sie bearbeitet werden (oder das Programm zum Abbrechen bringen).

Es ist eine Design-Entscheidung, wie solche Fehler behandelt werden. In unserem Fall haben wir uns entschieden, die zwingenden Fehler abzufangen, und in eine *RemoteException* verpackt an den Client zu schicken. Für unsere initiale Implementierung ist diese Methode am besten verwendbar, da vorhandene Java-Mittel benutzt werden können, um die Information über Fehler an den Client zu transportieren, wo sie ausgewertet werden können.

WSRF bietet für die Fehlerbehandlung auch eine Spezifikation an, die WS-BaseFaults [LiMe 06]. Diese können in WSDL erweitert und für jeden Fehlertyp einzeln designed werden. Durch den hohen Aufwand liegt dies außerhalb des Scope dieser Arbeit, bietet sich jedoch als ideale Weiterentwicklungsmöglichkeit an.

5.1.8 QNames – Namespaces für Variablen

Da wir einerseits intern über normale Java-Methoden auf Variablen zugreifen, andererseits diese auch teilweise über XML-Beschreibungen adressiert werden sollen, brauchen wir eine zentrale Stelle, an der solche Bezeichnungen verwaltet werden. Wir verwenden dazu *Qualified Names* (QNames), wie auszugsweise in Listing 5.1 gezeigt.

Listing 5.1: QNames.java (Auszug)

```
public interface VOQNames {
    public static final String NS =
        "http://www.cojocaru.de/Namespaces/VO/VOInstanceService";
    public static final QName RP_PERSONS = new QName(NS, "VOPersons");
    public static final QName RESOURCE_PROPERTIES = new QName(NS,
        "VOResourceProperties");
}
```

Unter einem *QName* ist ein Bezeichner [3] zu verstehen, der sich aus einem lokalen Bezeichner und einem optionalen vorangestellten Namensraumpräfix [4] zusammensetzt.

Die URL, die im Kontext von XML Namespace verwendet werden, sind zunächst nur reine Namen. Im Gegensatz zu normalen URL befinden sich dahinter keine (z.B. per Browser ladbaren) Dokumente. Allerdings hindert niemand die Urheber von Namespace URLs daran, tatsächlich Dokumente unter diesen URLs bereitzustellen.

[3] Name einer Variablen, Funktion, Klasse oder Methode. Je nach Programmiersprache gelten unterschiedliche Konventionen für Bezeichner.

[4] Ein Präfix ist eine vorangestellte Zeichenfolge.

5.2 Implementierung des Lebenszyklus

Eine Virtuelle Organisation besitzt in ihrem Lebenszyklus verschiedene Transitionen mit erkennbaren Eingangs- und Ausgangszuständen. Bei der *Erzeugung* geht die VO von „nicht-existierend“ zu „existierend“ über. Dabei besitzt die Virtuelle Organisation allerdings noch keinerlei Eigenschaften, sie ist eine leere Hülle. Die *Initialisierung* basiert auf dieser leeren VO und setzt eine erste, definierte Konfiguration. Da diese beiden Phasen stets aufeinander folgen, bietet es sich an, sie zu kombinieren und von den weiteren Phasen abzusetzen.

Im folgenden *Betrieb* bzw. *Anpassung*siterationen wird jeweils von einer konkreten Konfiguration in eine nächste übergegangen. Da dieser Wechsel den Hauptteil der Lebensspanne einer VO ausmacht, und sie funktionell gesehen gleich sind, lassen sich auch diese beiden Phasen kombinieren.

Um durch *Auflösung* in die Nichtexistenz überzugehen, benötigen wir nach dem Lebenszyklus-Modell eine Transition aus einer normalen Konfiguration. Danach folgt allerdings keine weitere Aktivität an dieser VO, es ist ein finaler Schritt. Daher werden wir diese Möglichkeit mit den beiden vorhergehenden Phasen, die ebenfalls auf einer normalen Konfiguration basieren, verbinden.

Um eine VO in einem Grid abzubilden, benötigen wir also zwei Web Services, durch die ein Verwalter die Übergänge auslösen kann. Wir verwenden daher das für diesen Fall geeignete Factory-Pattern, das bereits oben in 5.1.3 kurz erklärt wurde. Es beinhaltet einen Factory Service, der die Möglichkeit zur Erzeugung bietet. Dabei werden wir die Initialisierung auch direkt und implizit an die Erzeugung anknüpfen, so dass der Schritt für den Verwalter atomar ist. Ergebnis ist eine VO, die mit Hilfe eines zweiten Dienstes, dem Instance Service, bearbeitet werden kann. Hier werden sich alle Methoden finden, die notwendig sind, um Transitionen bezüglich des Betriebes, der Anpassung, sowie der Auflösung auszuführen.

5.3 Implementierung der VO im WSRF

Wie wir im Abschnitt 2.2 gelernt haben, ist der Kern des WSRF die WS-Resource. Diese besteht aus einem Web Service und, falls sie einen Zustand besitzen soll, aus einer Ressource. Der vorige Abschnitt hat gezeigt, dass wir in unserem Fall zwei Web Services implementieren. Der erste, der Factory Service, hat dabei keinen Zustand. Er wird aufgerufen und gibt ein Ergebnis zurück, auf das sich nicht mehr bezogen werden kann. Der Instance Service hingegen hat einen Zustand, da er die in 3.3.5 geforderten Eigenschaften einer VO speichern muss.

In diesem Abschnitt werden wir die Implementierung dieser beiden Dienste diskutieren. Dabei werden wir jeweils die Beschreibungen der Schnittstellen in WSDL betrachten. Die Umsetzung in Programmcode wird dann im nächsten Abschnitt gezeigt, wo die Implementierung in GT4 gezeigt wird.

Der entscheidende Vorteil dieser WSDL-Definition besteht nämlich darin, dass sie von der VO Implementierung selbst unabhängig ist. Die Verwalter können daher mit beliebig implementierten Clients auf die VO zugreifen. Auch bei Veränderungen innerhalb der Web Services, soweit sie nicht das Interface betreffen, bleibt der Aufrufer davon unberührt.

5.3.1 Factory Service

Damit ein Client überhaupt die Methoden des Factory Services in Anspruch nehmen kann, müssen diese in einer unabhängigen Beschreibungssprache dargestellt werden. Hierzu dient die in Kapitel 2 beschriebene Web Service Description Language (WSDL).

Um die Kompatibilität unseres Dienstes zu erhöhen, bedienen wir uns verschiedener Standarddefinitionen. Diese werden zu Beginn der Dienstbeschreibung eingebunden. Für die eigenen Definitionen wird dann ein neuer Namensraum unter einer URL angelegt, die von uns dann verwaltet wird. Die Namensraum-Importe und -Definitionen des genannten Services können in Listing 5.2 begutachtet werden. Im Vergleich zu den Definitionen des Instance Services, die wir später beleuchten werden, ist hier eine sehr knappe Struktur sichtbar.

Listing 5.2: VOFactoryService Namensraum Definitionen

```
<definitions name="VOFactoryService"
    targetNamespace="http://www.cojocaru.de/Namespaces/VO/VOFactoryService"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://www.cojocaru.de/Namespaces/VO/VOFactoryService"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

Neben dem dedizierten Namespace für den Factory Service, *VOFactoryService* benannt, wird hier das WSDL-Schema eingebunden. Dieses definiert die grundlegenden Strukturen einer auf XML aufbauenden WSDL Datei. Zusätzlich benötigen wir hier das Addressing-Schema, das Informationen über die Struktur einer Endpoint Reference enthält.

Listing 5.3 zeigt die für die Erzeugung einer VO verwendeten Datentypen. Das Element *createVO* ist dabei der Parameter, der mit der `createVO()`-Methode übergeben wird. In unserem Fall haben wir entschieden, hier einen String mitzugeben. Dieser entspricht dem Namen der VO, den wir als eindeutigen Identifikator verwenden.

Listing 5.3: VOFactoryService.wsdl (Ausschnitt: Datentypen)

```
<xsd:element name="createVO" type="xsd:string"/>
<xsd:element name="createVOResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="wsa:EndpointReference"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

Als Ergebnis des `createVO()`-Aufrufes haben wir eine Endpoint Reference definiert. Diese beinhaltet die Adressierungsinformation des Instanz Dienstes, sowie einen Identifikator für die eben erzeugte Ressource.

Da der Typ der Endpoint Reference in dem Addressing-Schema definiert ist, und wir diesen in unserer Beschreibung auch nutzen wollen, muss er über ein ⟨`import`⟩-Tag eingebunden werden. Dies wird in Listing 5.4 gezeigt.

Listing 5.4: VOFactoryService.wsdl (Ausschnitt: Import des WS-Addressing)

```
<xsd:import
   namespace="http://schemas.xmlsoap.org/ws/2004/03/addressing"
   schemaLocation=
      "/usr/local/globus-4.0.1/share/schema/ws/addressing/WS-Addressing.xsd
/>
```

Nun, da wir die Datentypen fest gelegt haben, können wir sie einer jeweiligen Nachricht zuordnen. Diese Zuordnung wird in Listing 5.5 gezeigt. Dabei gibt es für jede Anfrage und jede Antwort je eine Nachricht, hier *CreateVORequest* und *CreateVOResponse*, denen die oben genannten Datentypen aus unserem Namespace zugeordnet werden.

Listing 5.5: VOFactoryService.wsdl (Ausschnitt: Nachrichten)

```
<message name="CreateVORequest">
    <part name="request" element="tns:createVO"/>
</message>
<message name="CreateVOResponse">
        <part name="response" element="tns:createVOResponse"/>
</message>
```

Zum Schluss müssen wir noch den PortType definieren. Dieser beschreibt eine aufrufbare Operation und verknüpft sie mit den entsprechenden Eingangs- und Ausgangsnachrichten, wie in 5.6 dargestellt.

Listing 5.6: VOFactoryService.wsdl (Ausschnitt: PortType)

```
<portType name="VOFactoryServicePortType">
    <operation name="createVO">
        <input message="tns:CreateVORequest"/>
        <output message="tns:CreateVOResponse"/>
    </operation>
</portType>
```

Als Ergebnis des `createVO()`-Aufrufes erhält der Client also eine Endpoint Reference. Über diese kann er den Instance Service auffinden und Operationen auf der gewünschten VO ausführen. Einen beispielhaften Ausschnitt aus einer EPR-Datei zeigt das Listing 5.7.

Listing 5.7: „Endpoint Reference (Ausschnitt) “

```
        ...
<ns2:Address xsi:type="ns2:AttributedURI">http://192.168.218.21:8080/wsrf
     /services/cojocaru/VO/VOInstanceService</ns2:Address>
 <ns2:ReferenceProperties xsi:type="ns2:ReferencePropertiesType">
     <ns1:VOResourceKey
   xmlns:ns1="http://www.cojocaru.de/Namespaces/VO/VOInstanceService_instance">
     1386211</ns1:VOResourceKey>
 </ns2:ReferenceProperties>
        ...
```

Wie man in diesem Listing sehen kann, enthält der EPR neben der URI des Instance Services den vorhin erwähnten *ResourceKey* (auch ResourceID genannt, hier 1386211) [GHR 06] der erzeugten bzw. abgerufenen Virtuellen Organisation. Über dieses Informationspaar kann der Client nun die von ihm adressierten VO verwalten.

5.3.2 Instance Service

Der Instanz Dienst hat neben der Aufgabe, die Operationen dem Client zur Verfügung zu stellen, noch eine weitere Funktion. Er beschreibt nämlich die Abbildung der Attribute des VO-Modells 1.1 auf die WS-Resource. Dabei werden wir gleich sehen, dass die einzelnen Komponenten im VO-Modell, wie Personen, Rollen, Dienste oder Ressourcen, die wir ja bereits in Abschnitt 3.3.5 als Properties gefordert haben, nun umgesetzt sind.

Als Ergebnis des `createVO()`-Aufrufes bei dem oben beschriebenen Factory Service haben wir eine Endpoint Reference erhalten. Diese ist die eindeutige Adressbeschreibung unserer als WS-Resource implementierten VO.

Um nun Transitionen des Betriebes, der Anpassung und der Auflösung vorzunehmen, haben wir uns entschieden, einen separaten Dienst zu schreiben. Dieser Dienst besitzt eine eigene URL, die in der erhaltenen Endpoint Reference bereits gespeichert ist.

Listing 5.8: VOInstanceService.wsdl (Ausschnitt: Definitionen)

```
<definitions
    name="VOInstanceService"
    targetNamespace="http://www.cojocaru.de/Namespaces/VO/VOInstanceService
    _instance"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://www.cojocaru.de/Namespaces/VO/VOInstanceService_instance"
    xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
         ResourceProperties-1.2-draft-01.xsd"
    xmlns:wsrpw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
         ResourceProperties-1.2-draft-01.wsdl"
```

```
    xmlns:wsrlw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
         ResourceLifetime-1.2-draft-01.wsdl"
    xmlns:wsrl="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
         ResourceLifetime-1.2-draft-01.xsd"
    xmlns:wsdlpp="http://www.globus.org/namespaces/2004/10/WSDLPreprocessor"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
>
<wsdl:import namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-
                    WS-ResourceProperties-1.2-draft-01.wsdl"
    location="../../../wsrf/properties/WS-ResourceProperties.wsdl" />
<wsdl:import namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-
                    WS-ResourceLifetime-1.2-draft-01.wsdl"
    location="../../../wsrf/lifetime/WS-ResourceLifetime.wsdl" />
```

Die Kommunikationsmöglichkeiten mit dem Instance Service sind wieder in der WSDL-Datei des Instance Services beschrieben. Diese ist nun ungleich ausführlicher als beim Factory Service. Bereits die Definition der Namespaces ist länger. Hier in Listing 5.8 erkennen wir, dass wir auf dem WSRF basieren. Es werden nämlich die Schemata für WS-ResourceProperties eingebunden, die die Struktur beschreiben, nach der sich die Eigenschaften einer WSRF-konformen VO richten muss [GrTr 06].

In dem genannten Listing finden wir eine Referenz auf die WS-ResourceLifetime. Im letzten Abschnitt haben wir bereits beschlossen, dass die Transition zur Auflösung der VO ebenfalls vom Instance Service behandelt werden soll. Die WS-ResourceLifetime spezifiziert nun die Methoden und Prozesse zur sofortigen oder geplanten Auflösung, so dass wir damit die gewünschte Transition sogar in zwei Varianten implementieren [SrBa 06].

Eine Auffälligkeit ist noch die Einbindung des WSDL-Präprozessor [Soto 05]. Dieser beinhaltet Informationen, die uns eine erhebliche Arbeitserleichterung bringen. Damit haben wir hier nun aufgrund unserer Einbettung in das standardisierte WSRF die Möglichkeit, bereits vordefinierte Nachrichtentypen und Operationen mit geringem Aufwand einzubinden. Die überwiegende Tätigkeit entfällt dann auf das Definieren der Eigenschaften der VO, da diese ja mittels der WSRF-Methoden manipuliert werden sollen.

Listing 5.9: VOInstanceService.wsdl (Ausschnitt: PortType)

```
<portType
    name="VOInstanceServicePortType"
    wsrp:ResourceProperties="tns:VOResourceProperties"
    wsdlpp:extends="wsrpw:GetResourceProperty
                    wsrpw:GetMultipleResourceProperties
                    wsrpw:SetResourceProperties
                    wsrpw:QueryResourceProperties
                    wsrlw:ScheduledResourceTermination
                    wsrlw:ImmediateResourceTermination
                    ">
</portType>
```

Die Einbindung der Manipulationsmethoden des WSRF erfolgt, wie in Listing des PortTypes 5.9 dargestellt, durch einfache Angabe des Methodennamens. Dies ist, wie am Import des Schemas zu erkennen, eine Hilfestellung des Globus Toolkit. Ohne diesen Präprozessor müssten die entsprechenden Operationen und Nachrichtentypen selbst definiert werden.

Listing 5.10: VOInstanceService_flattened.wsdl (Ausschnitt: Operation GetResourceProperty)

```
<wsdl:operation name="GetResourceProperty">
    <wsdl:input name="GetResourcePropertyRequest"
        message="wsrpw:GetResourcePropertyRequest"
        wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/
            wsrf-WS-ResourceProperties/GetResourceProperty"/>
    <wsdl:output name="GetResourcePropertyResponse"
        message="wsrpw:GetResourcePropertyResponse"
        wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/w
```

```
            srf-WS-ResourceProperties/GetResourcePropertyResponse"/>
    <wsdl:fault name="InvalidResourcePropertyQNameFault"
        message="wsrpw:InvalidResourcePropertyQNameFault"/>
    <wsdl:fault name="ResourceUnknownFault" message="wsrpw:ResourceUnknownFault"/>
</wsdl:operation>
```

Globus verarbeitet diese WSDL und erzeugt eine „flattened“ Version, die an Stelle dieser Präprozessor-Befehle die tatsächlich notwendigen Informationen enthält. Das Ergebnis ist in Listing 5.10 dargestellt.

In der WSDL-Beschreibung des Interfaces des Instance Services findet sich außerdem eine Beschreibung der Eigenschaften der von uns implementierten Virtuellen Organisation. Zwar halten wir diese nicht in dem Web Service selbst, sondern in der Ressource, allerdings muss für die korrekte Übertragung der Informationen deren Serialisierung abgestimmt sein. Da die Eigenschaften beim Manipulieren zwischen Client und Web Service übertragen werden müssen, werden sie hier, wie in 5.11 ausschnittweise gezeigt, mit ihren Datentypen und damit implizierten Serialisierungen definiert.

Listing 5.11: VOInstanceService.wsdl (Ausschnitt: ResourceProperties)

```
<types>
    <xsd:schema
        targetNamespace=
            "http://www.cojocaru.de/Namespaces/VO/VOInstanceService_instance">
        <xsd:element name="VOPersons">
            <xsd:complexType name="VOPersonArray">
                <xsd:sequence>
                    <xsd:element
                        ref="tns:VOPerson" minOccurs="0" maxOccurs="unbounded"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name="VOPerson">
            <xsd:complexType name="VOPersonProperties">
                <xsd:sequence>
                    <xsd:element name="VOPersonFirma" type="xsd:string"/>
                    <xsd:element name="VOPersonName" type="xsd:string"/>
                    <xsd:element name="VOPersonOrt" type="xsd:string"/>
                    <xsd:element name="VOPersonPlz" type="xsd:string"/>
                    <xsd:element name="VOPersonPosition" type="xsd:string"/>
                    <xsd:element name="VOPersonPrincipal" type="xsd:string"/>
                    <xsd:element name="VOPersonStrasse" type="xsd:string"/>
                    <xsd:element name="VOPersonVorname" type="xsd:string"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name="VOResourceProperties">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element ref="wsrl:CurrentTime"/>
                    <xsd:element ref="wsrl:TerminationTime"/>
                    <xsd:element ref="tns:VOMemberVO"/>
                    <xsd:element ref="tns:VOName"/>
                    <xsd:element ref="tns:VOPersons"/>
                    <xsd:element ref="tns:VOResources"/>
                    <xsd:element ref="tns:VOGroups"/>
                    <xsd:element ref="tns:VORights"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
</xsd:schema>
```

In Listing 5.11 sehen wir einen beispielhaften Auszug aus der Beschreibung der ResourceProperties. Unter

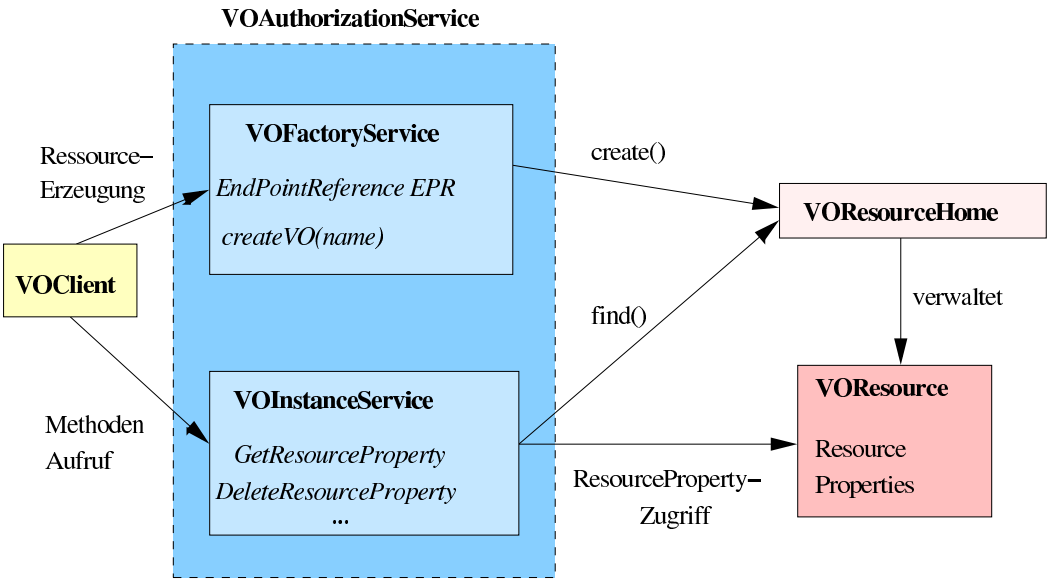


Abbildung 5.1: Anwendung des Factory-Pattern zur Erzeugung von VOs [SoCh 06]

Service	Bezeichnung	Erläuterungen
Factory Service	VOFactoryService	Erzeugung der VO
Instance Service	VOInstanceService	Manipulation der VO
Ressource	VOResource	Kapselung der Eigenschaften einer VO
Ressource Home	VOResourceHome	Sammlung aller VOResource-Objekten
Policy Decision Point	VOMemberPermit	Autorisierung von Zugriffen
Policy Information Point	VOMemberInfo	Informationssammlung für VOMemberPermit

Tabelle 5.1: Zusammenhang zwischen Komponenten und Klassennamen unserer Implementation

dem Element mit dem Namen „VOResourceProperties“ beginnt die Beschreibung des Resource Property Documents.

Wie man sehen kann, hat dieses Dokument acht Kindelemente unterhalb des Anfangs. Diese entsprechen den Attributklassen im VO-Modell. Sie enthalten wiederum komplexere Strukturen, hier gezeigt anhand der Personen, welche die Detailinformation aufnehmen.

Die Datenstruktur `VOPersons` enthält als Unterelement einen Array (`VOPersonArray`) von `VOPerson`. Die `VOPerson` selbst wiederum enthält eine Sequenz aus Strings, die einige Informationen über die jeweilige Person beherbergen können. Welche Informationen tatsächlich für eine Person gespeichert werden können, ist hier relativ anpassbar.

5.4 Implementierung der VO in GT4

Im vorigen Abschnitt haben wir uns die Schnittstellendefinitionen der Web Services angesehen, wie sie durch die Implementation der Ressource im WSRF aussehen müssen. Nun werden wir auf Basis der bisher gewonnenen Erkenntnisse die tatsächliche Programmierung beschreiben.

Eingangs möchten wir kurz alle von uns geschriebenen Komponenten vorstellen, bevor diese im Detail erläutert werden. Die Bezeichnung des Factory- und Instance Services waren oben schon in den WSDL-Beschreibungen erwähnt. In Tabelle 5.4 folgt eine Übersicht über die implementierten Services mit den entsprechenden Erläuterungen, die im Laufe des Kapitels ausführlich erweitert werden.

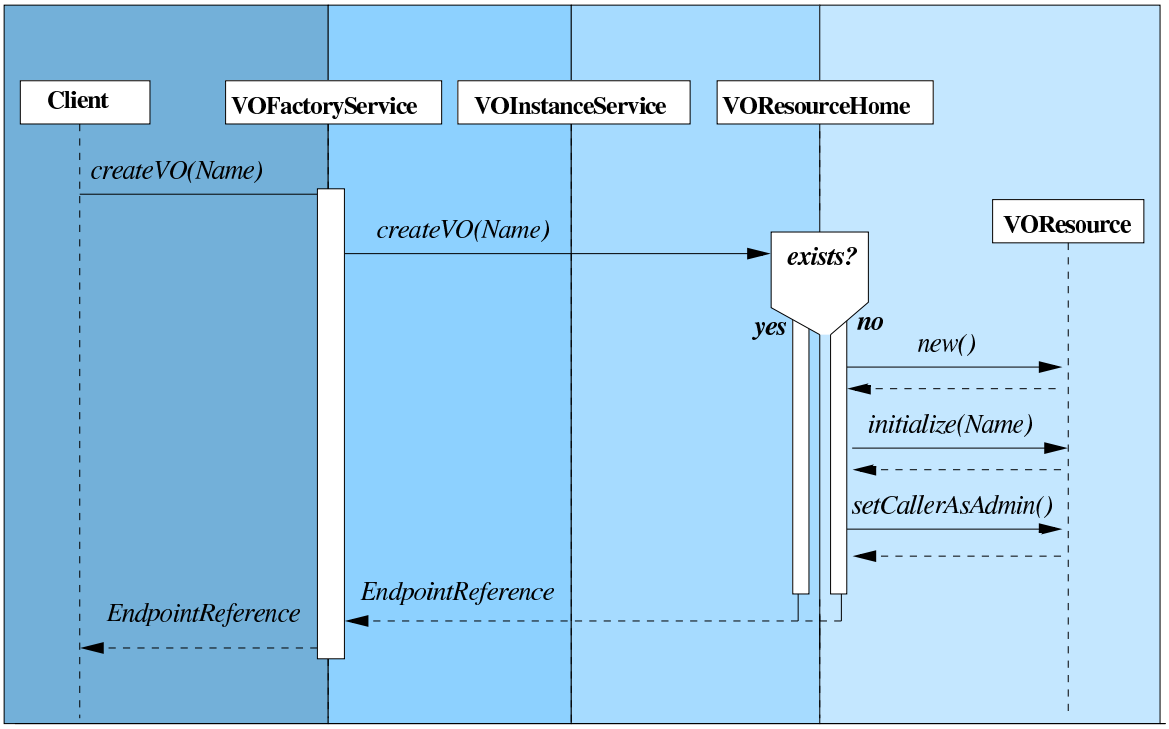


Abbildung 5.2: Nachrichtenfluss bei der Erstellung einer VO-Ressource

Eine graphische Übersicht über diese Komponenten zeigt Abbildung 5.1. Dabei sind die Klassen VOMemberPermit und VOMemberInfo zu dem VO AuthorizationService zusammengefasst, da sie nur als Paar existieren. In Abbildung 5.10 werden alle entwickelten Klassen in einem größeren Zusammenhang gezeigt, sowie der Kommunikationsfluss untereinander angedeutet.

Der *VOFactoryService* bietet eine Funktion zur Erzeugung einer VO. Dabei sucht er zunächst im VOResourceHome, ob bereits eine VO mit diesem Namen existiert. Falls nicht, erzeugt er eine neue Instanz einer *VOResource*, sonst fordert er die benannte an. Dannach gibt er die Referenz auf die WS-Resource, mit URL zum VOInstanceService und ID der instantiierten VOResource, zurückt. Der *VOInstanceService* enthält alle benötigten Methoden, um die Konfiguration der VO zu manipulieren oder diese aufzulösen. Für diese Arbeit werden die Methoden und ihre Funktionalitäten vom WSRF spezifiziert.

Im folgenden werden wir uns den einzelnen erstellten Komponenten genauer widmen.

5.4.1 VOFactoryService

Wie der Name „Factory-Pattern“ schon andeutet, benötigen wir für die Implementierung dieses Patterns eine Fabrik-Klasse. Deren Aufgabe ist es, Objekte nach einem vorgeschriebenem Verfahren zu erzeugen. Außerdem werden in unserem Fall die erzeugten Objekte in einem Register eingetragen, um späteres erneutes Aufrufen zu ermöglichen, und mit einer festgelegten initialen Konfiguration zu versehen. Wir folgen damit dem Rat des GT4-Tutorials [5]. Wie oben beschrieben, wird durch den Einsatz einer Factory die Erzeugung und Initialisierung für den Verwalter atomar. Beide Phasen werden in einem Schritt durchlaufen. Die im Hintergrund auf der Grid-Seite ablaufenden Prozesse werden im Message Sequence Chart (MSC) 5.2 dargestellt. Der folgende Ablauf wird für den Client transparent:

1. *VOFactoryService*: Weiterleitung des `createVO(Name)`-Aufrufes an das VOResourceHome zusammen mit dem *Principal* des Aufrufers.

[5] Weitere Informationen unter http://gdp.globus.org/gt4-tutorial/

2. *VOResourceHome*: Überprüfen, ob es bereits eine VO mit diesem Namen gibt.
   Falls *ja*:
   - Rückgabe des existierenden ResourceKeys (ResourceIDs) 5.1.2.

   Falls *nein*:
   - Erzeugen einer neuen Ressourcen-Instanz.
   - Initialisieren mit dem gegebenen Namen.
   - Erzeugen einer Administratorengruppe mit Berechtigung auf alle Operationen. Mitglied ist der Aufrufer.
   - Rückgabe des neuen ResourceKeys.
3. *VOFactoryService*: Erzeugen eines EPR und Rückgabe an den CLient.

In unserer Implementierung bietet der VOFactoryService zusätzlich zur Erzeugung noch eine Information über die bestehenden VOs an. Die Methode `getListVO()` liefert dabei einen Array aus Strings zurück. Wie oben erwähnt, ist der Name der VO ein eindeutiger Bezeichner. Eine besondere Funktion des VOFactoryServices ist es, beim Aufruf der `createVO()`-Methode das Register der VOs nach einer bereits bestehenden VO mit diesem Namen zu durchsuchen. Existiert diese VO bereits, wird eine Referenz auf die existierende VO zurück gegeben. Dadurch wird es den Verwaltern ermöglicht, aus einer Liste die zu bearbeitende VO auszuwählen und einen EPR zu erhalten, ohne diesen ständig zwischenspeichern zu müssen.

5.4.2 VOInstanceService

Dem Instance Service kommt eine zentrale Rolle in der Verwaltung Virtueller Organisationen zu. Durch die Implementation im WSRF mittels GT4 haben wir die Möglichkeit, erhebliche Teile standardisierter Funktionen einzubinden. Das Globus Toolkit stellt nämlich für die WSRF-Operationen nicht nur den in 5.3.2 beschriebenen WSDL-Präprozessor, sondern auch gleich die Implementation der Methoden in Form von Providern zur Verfügung. Diese Provider, einer je Methode, werden dem Instance Service hinzugefügt. Dadurch ersparen wir uns einerseits die Schreibarbeit, schützen uns andererseits aber auch vor fehlerhafter Implementierung der standardisierten Methoden.

Durch das Importieren der Methoden in die WSDL und durch das Beschreiben der Resource Properties kann der Operation Provider entsprechend konfiguriert werden, und die gewählten Methoden stehen automatisch zur Verfügung.

Im Sequenz-Diagramm 5.3 wird der Nachrichtenfluss bei einem Client-Aufruf (Invocation) dargestellt. Will der Client beispielsweise die Methode `getResourceProperty(X)` auf einer VO ausführen, um die Liste der enthaltenen WS-Resourcen zu erhalten, stellt er die Anfrage an den VOInstanceService. Dieser sucht dann im VOResourceHome nach der referenzierten VOResource. Findet er sie, ruft er die entsprechende ResourceProperty ab und schickt sie an den Client zurück. Diese Logik enthält komplett der Provider für die `getResourceProperty()`-Methode, genauer der `GetResourcePropertyProvider`, der sich in der Globus-WSRF-Implementation im Java Packet *org.globus.wsrf.impl.properties* befindet.

5.4.3 VOResource

Zu einer WS-Resource verbunden mit dem VOInstanceService ist die VOResource. Diese Klasse beinhaltet die Informationen über den Zustand der Virtuellen Organisation. Alle diese Eigenschaften werden in Form von Attribute (siehe auch Abschnitt 2.3) bereitgestellt. Beschrieben und damit für einen Aufrufer zugreifbar werden sie in der WSDL-Datei des Instanz-Dienstes, gezeigt in 5.3.2. Dieser ermöglicht dann, wie wir im vorigen Abschnitt gesehen haben, den Zugriff auf die Ressource.

Das VO Resource Property Diagramm 5.4 stellt nicht nur die Eigenschaften einer VO, sondern auch die Attributen, die jeder von uns definierten Property zugeordnet sind, schematisch dar. Die VOPersons-Property

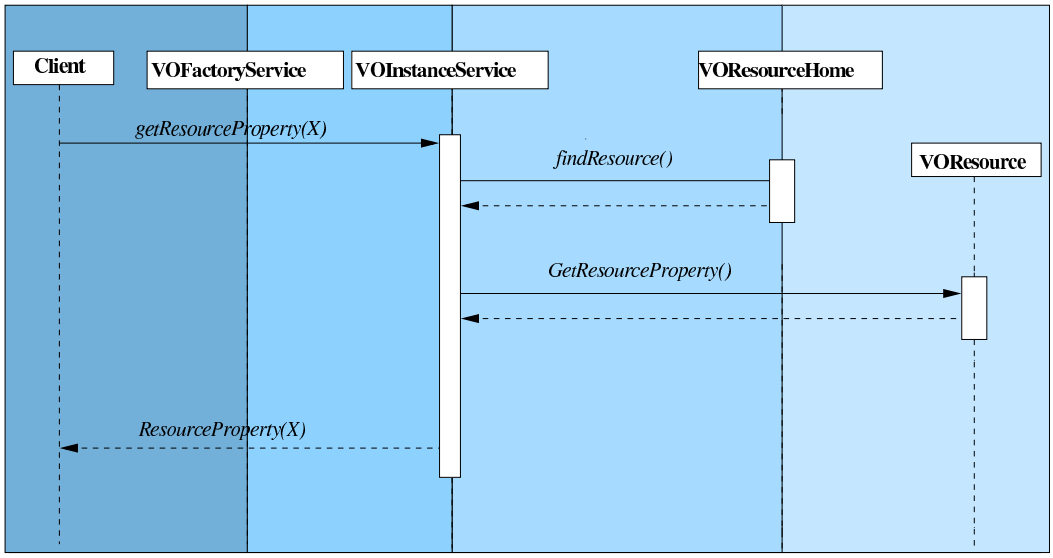


Abbildung 5.3: Sequenz-Diagramm zum Aufruf einer VO-Ressource [SoCh 06]

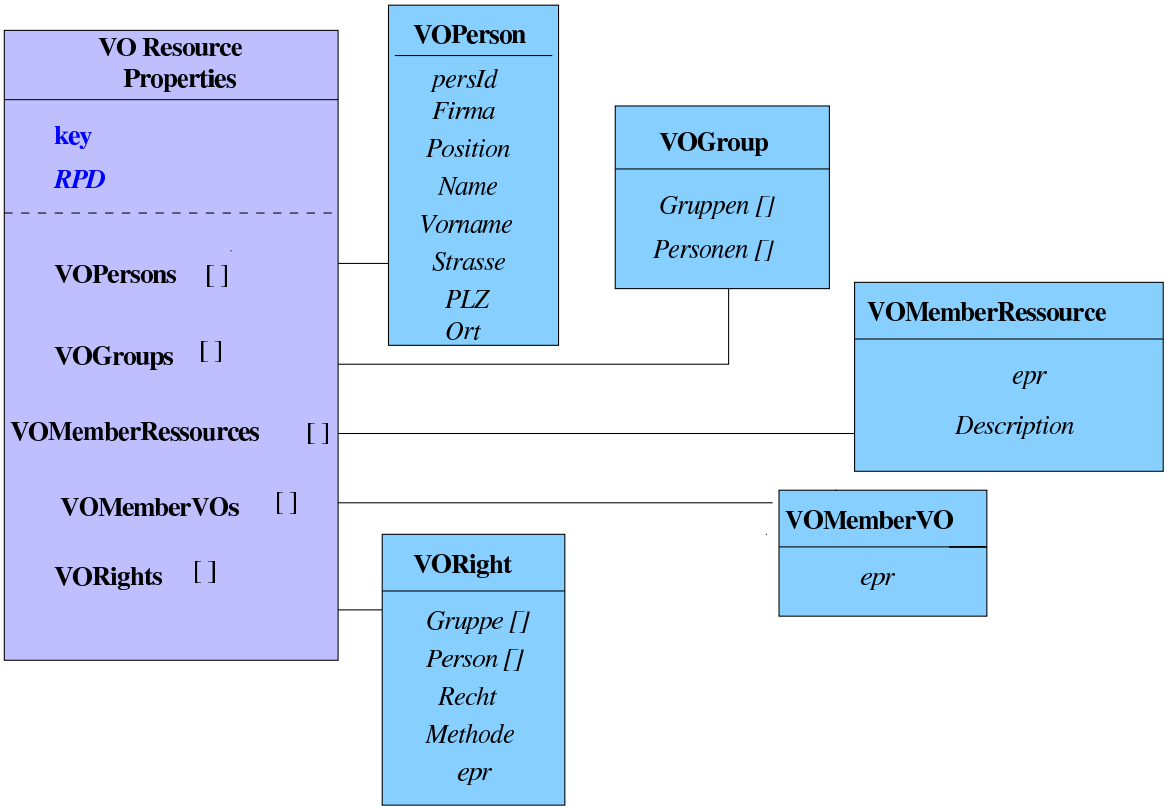


Abbildung 5.4: Resource Property Diagramm (Beispiel)

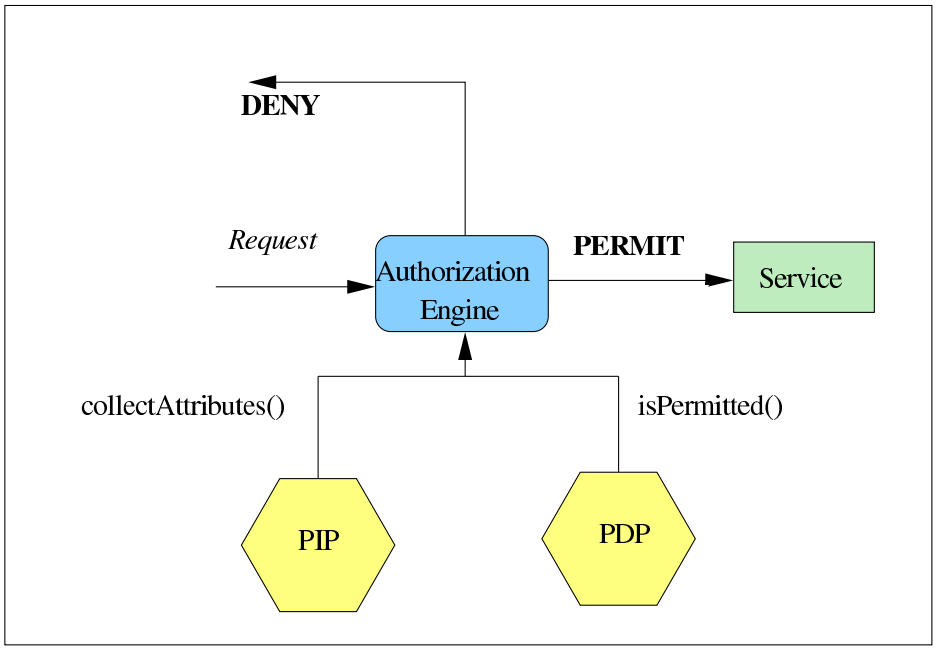


Abbildung 5.5: PDP und PIP Chain [FrAn 05]

besteht aus mehreren Elementen vom Typ VOPerson. Eine VOPerson-Eigenschaft wird weiterhin durch mehrere Attribute, wie Name, Vorname, Ort, PLZ etc. beschrieben.

In der Abbildung 5.9 werden die vom Globus Build Service aus unserer WSDL-Beschreibung der Resource-Properties erzeugten Klassen dargestellt. Die VOResourceProperties beinhalten die Eigenschaften, die unsere VO beschreiben, als Member Variablen. Zum Beispiel werden die in der VO enthaltenen WS-Resourcen hier in einer Variable VOMemberRessources gekapselt, die ein Array von VOMemberRessource ist. Letztere wiederum speichert die Endpoint Reference der in der VO enthaltenen WS-Resource, sowie eine textuelle Beschreibung der Ressource. VOPersons kapselt ein Array von VOPerson-Objekten, die die auf Rollen in unserer VO abgebildeten Personen speichern. Die Beziehungen und weiteren Properties sind in der genannten Abbildung zu sehen.

Personen können zu Gruppen zusammengefasst werden. Eine Gruppe kann einer übergeordneten Gruppe hinzugefügt werden. Dies stellt unsere Implementierung des Rollenkonzeptes einer Virtuellen Organisation dar.

Eine VOMemberRessource ist eine WS-Ressorce, die von den VO Zugehörigen aufgerufen wird und sich durch eine EPR identifiziert. Die VOMemberVO ist eine weitere Property der Ressource. Sie stellt eine Sub-VO, die der übergeordneten Virtuellen Organisation zugehört, dar. Eine oder mehrere VOMemberResource können zu einem Array VOMemberResources hinzugefügt werden.

Durch die VORights werden die Zugriffsrechte auf Ressourcen (VOMemberResources) unter Kontrolle gehalten.

Da wir auch WSRF-basierte Ressourcen beschreiben wollen, ist es notwendig, der Anforderung des Standards zu folgen. Das Framework schreibt folgendes vor: jede WS-Resource muss eine RessourceID für ihre Identität verwenden und jede Ressource muss ihre Properties in einem eigenen Resource Property Document speichern (siehe auch 5.1.2).

In unserem Ansatz wird auf die von uns erzeugten VO-Ressourcen mit Hilfe von EPRs zugegriffen, die nicht anders als aus einer URI unseres VOInstanceServices und der ID der VO-Ressource besteht. Außerdem beinhaltet die WSDL-Beschreibung vom Instanz-Dienst die XML-Struktur des RPDs.

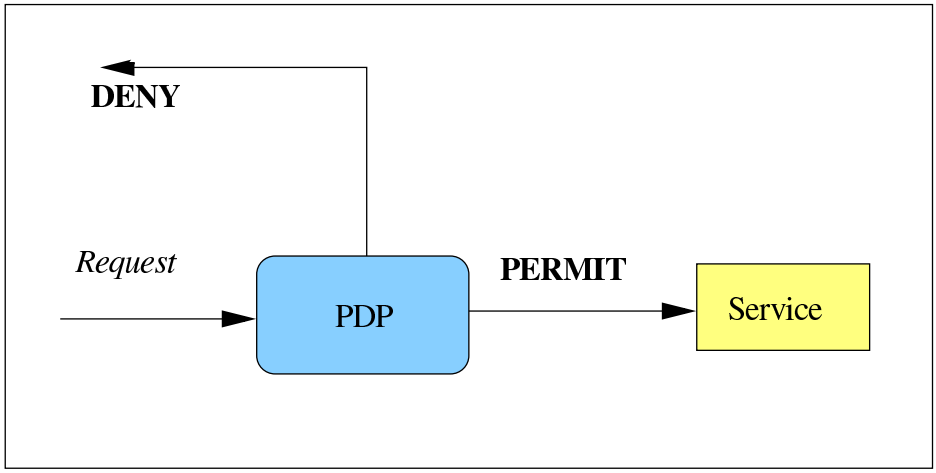


Abbildung 5.6: PDP Auswertung [FrAn 05]

5.4.4 VOMemberInfo

Die Sicherheitsmaßnahmen spielen eine bedeutende Rolle im Bezug auf VO Bildung und Zugriff auf VO-Ressourcen. Im Rahmen unserer Arbeit betrachten wir die Autorisierung, die Authentifizierung jedoch nicht. Die Annahme dabei ist, dass jeder der sich gegenüber Grid authentisiert, auch eine VO anlegen darf, sobald keine zusätzlich eingebauten Sicherheitsmechanismen dies einschränken.

In unserer Arbeit wird die Autorisierung auf Web Services (z.B. VOFactory-und VOInstanceService) definiert. Für jeden Service wird je ein Security Descriptor angelegt, der die WS-Autorisierungsregel beinhaltet. Im Security Descriptor werden dafür sogenannte Policy Decision Point (PDP)-Ketten ausgewertet, um weiterhin eine Zugriffs-Entscheidung zu treffen.

Ein PDP kann also auch als ein „Entscheidungsknoten“ gesehen werden. Für eine PDP-Kette wird die Reihenfolge der Ausführung der PDPs festgeslegt. Jede einzelne PDP-Entscheidung beeinflusst die „Endentscheidung“. Bei der Ausführung eines Clients wird dessen Zertifikat auf Autorisierung geprüft; beim Starten des Globus-Containers werden weiterhin in der `server-deploy.wsdd`-Datei Informationen bezüglich der durchzuführenden PDPs geholt (Siehe auch Abbildung 5.7).

Im Globus Toolkit 4 kann eine Kette von PDPs und Policy Information Points (PIPs) definiert werden. Wie die Namen schon sagen, trennt sich hier die Aufgabe der Klassen in Informationsgewinnung und Entscheidung auf Basis der gewonnenen Information. In der Abbildung 5.5 wird die Ausführung einer Kette von PDPs und PIPs dargestellt.

5.4.5 VOMemberPermit

Nachdem alle Rechte und Gruppenzugehörigkeiten des Aufrufers zusammen getragen wurden, fällt der PDP[6] auf deren Basis eine positive oder negative Entscheidung. In der Abbildung 5.6 wird graphisch dargestellt, wie die PDP-Auswertung statt findet [SFA 07].

Im GT4 werden können mehrere PDPs und PIPs zu einer Authorization Chain verbunden werden. Die einzelnen Punkte werden dabei nacheinander evaluiert, die PIPs sammeln die Informationen und die PDPs treffen ihre jeweils eigene Entscheidung. Für das Endergebnis können dabei zwei unterschiedliche Strategien verwendet werden: `deny override` oder `permit override`.

Während der Auswertung von PDPs werden unterschiedliche Einzelentscheidungen getroffen. Dabei beeinflussen unterschiedliche Informationen, aber auch die Reihenfolge der PIPs, das Ergebnis. Wenn mindestens

[6] Auch als *interceptor* bezeichnet

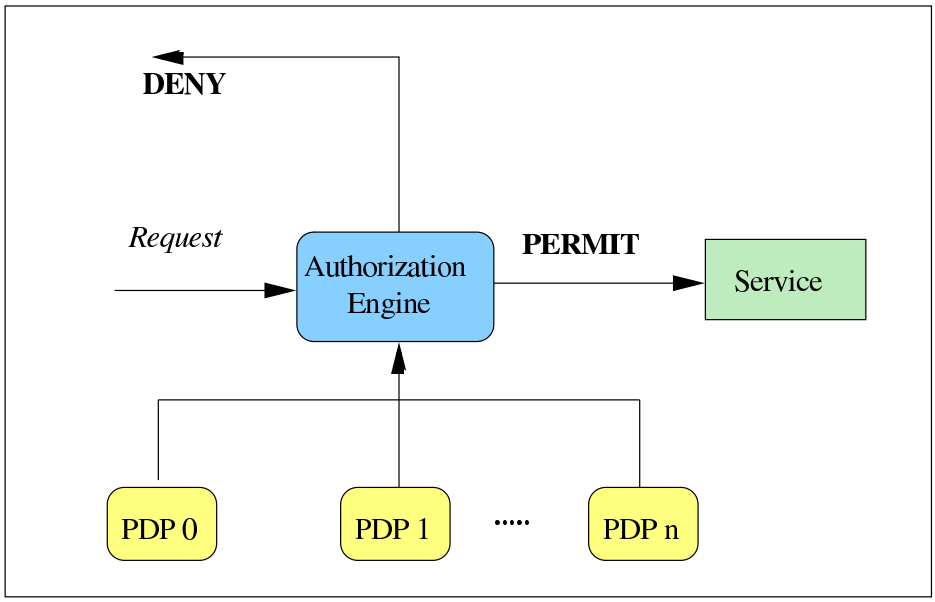


Abbildung 5.7: PDP Chain [FrAn 05]

ein PDP als `deny` ausgewertet wird, bei der Implementierung der Methode `deny override`, wird die Gesamtentscheidung auf `deny` gesetzt. Ebenso kann bei `permit override` ein einziges `permit` das Endergebnis auf Positiv übersteuern. Die Funktionsweise einer solchen Authorization Chain ist in 5.7 dargestellt.

Unser PDP vergleicht die Endpoint Reference der aufgerufenen WS-Resource mit denen, auf die der Aufrufer Berechtigungen besitzt. Diese Informationen erhält er von dem PIP (VOMemberInfo). Wird eine gefunden, muss noch das Recht auf die Methode geprüft werden. Dabei haben wir die Möglichkeit vorgesehen, ein Template mittels Asterisk anzugeben, ähnlich der Angabe, wie sie im Security Descriptor möglich ist. Eine Angabe „get*“ ermöglicht dann die Verwendung aller get-Methoden, also lesenden Zugriff. Der Erzeuger einer VO wird initial über die Administratorengruppe so auf alle Methoden der VO berechtigt („*“).

5.4.6 VOClient

Um die Implementation der Virtuellen Organisationen als WS-Ressourcen zu testen, haben wir eine einfache Kommandozeilenapplikation entwickelt. Diese kann eine VO erzeugen lassen und einfache Zustandstransitionen durchführen. Dadurch kann sie als Management-Software eingesetzt, wenn auch nicht mit viel Komfort oder Funktionsumfang. Wir haben uns lediglich darauf beschränkt, die Resource Properties bearbeiten zu können. Im Folgenden sehen wir uns die Arbeitsweise und Aspekte des Clients genauer an.

Der Client benötigt zu Beginn die URI des VOFactoryServices, um eine neue VO anlegen zu können. Dieses initiale Wissen kann entweder vom Verwalter konfiguriert werden, oder in der Client-Softare hart codiert werden. In unserem Fall werden wir es als Parameter an den Client übergeben.

Aus dieser URI kann der Client sich eine Endpoint Reference erzeugen, die in diesem Fall keine ResourceID enthält, da der Factory Service ja zustanslos ist.

Wie wir in der WSDL-Beschreibung oben gesehen haben, definiert der PortType eines Web Service, welche Operationen auf ihm ausgeführt werden können. Um das PortType-Objekt zu erhalten, gibt es für jeden Service einen Locator. Globus erzeugt uns die entsprechenden Klassen automatisch beim Übersetzen der Web Services. Mit Hilfe des Locators erhalten wir also den PortType des VOFactoryServices, auf dem wir die Methode zur Erzeugung einer VO ausführen können. Der Rückgabewert ist wiederum eine Endpoint Reference, diesmal mit der URI des VOInstanceSerivces und der ObjektID der erzeugten VOResource. Nun können wir über einen zweiten Locator anhand der erhaltenen Endpoint Reference den PortType der WS-Resource erhalten, die unsere VO darstellt.

Auf diesem PortType können wir im weiteren Verlauf alle Operationen ausführen, die Transitionen gemäß der Phasen Betrieb, Anpassung und Auflösung einer VO ermöglichen. Dabei dreht es sich hauptsächlich um die Manipulation der Resource Properties der VOResource.

Die Eigenschaften einer WS-Resource werden auf Wunsch in einem Resource Property Document dargestellt. Betrachtet man dieses Dokument, so hat es aufgrund seiner XML-Beschreibung eine Baumstruktur. Der Client besteht aus einer Reihe an statischen Methoden, eine für jeden Knoten im Baum. Zusätzlich gibt es verständlicherweise eine Ebene, in der die VO selbst ausgewählt (oder erzeugt) werden kann.

Dem Benutzer wird eine Information gegeben, auf welcher Ebene er sich momentan befindet. Dann erscheint eine nummerierte Liste mit Elementen, die die Informationen dieses Knotens darstellen. Im Weiteren erscheint eine Eingabeaufforderung mit den möglichen Commandos. Diese bestehen aus Wortbefehlen, die eventuell von einer Zahl gefolgt wird, die die zu bearbeitende Position der gezeigten Liste angibt.

5.5 Implementierung der VO

Bisher haben wir die Teile der Implementation betrachtet, die sich auf das WSRF und auf die durch das GT4 vorgegebenen Strukturen bezogen haben. Nun widmen wir uns der konkreten Programmierung. In diesem Abschnitt werden beispielhafte Code-Teile gezeigt und erläutert. Dabei wird stets auf die vorangegangenen Abschnitte bezug genommen.

5.5.1 VOFactoryService

Die Implementierung des *VOFactoryService*s ist relativ straightforward, da unser Service nur die zwei Funktionen `createVO(String name)`, die als Rückgabe ein EPR zurück gibt, sowie `getVOList()`, die ein Array von String mit den Namen aller existierenden VOs retourniert, besitzt.

Der Code ist ähnlich knapp wie die WSDL-Interfacebeschreibung. Sie besteht aus der bereits besprochenen `createVO()`-Methode, der Methode `getVOList()`, sowie einer `getInstanceVOResourceHome()`-Hilfsmethode , die das Resource Home, also die zentrale Sammelstelle für VOs, herausfindet.

Diese drei Methoden sind in auf das wesentliche reduzierter Form in Listing 5.12 dargestellt. Zu bemerken ist, dass die Methoden jeweils eventuelle Fehler abfangen und als RemoteException an den Aufrufer weiterleiten. Dies wurde aus Platzgründen hier herausgekürzt, ist aber mit den restlichen erstellten Dateien auf der beiliegenden CDROM zu finden.

Listing 5.12: VOFactoryService.java (gekürzt)

```
public class VOFactoryService {
    public CreateVOResponse createVO(String name) throws RemoteException {
        VOResourceHome home = getInstanceVOResourceHome();
        ResourceKey key = home.create(name,
            SecurityManager.getManager().getCallerPrincipal().toString(),
            instanceURI);
        EndpointReferenceType epr = null;
        epr = AddressingUtils.createEndpointReference(instanceURI, key);
        CreateVOResponse response = new CreateVOResponse();
        response.setEndpointReference(epr);
        return response;
    }
    protected VOResourceHome getInstanceVOResourceHome()
            throws NoResourceHomeException, ResourceContextException {
        Context initialContext = new InitialContext();
        ResourceContext ctx = ResourceContext.getResourceContext();
        String homeLoc = Constants.JNDI_SERVICES_BASE_NAME
            + ctx.getService() + "/instanceHome";
        return (VOResourceHome) initialContext.lookup(homeLoc);
    }
```

```
    public GetVOListResponse getVOList(GetVOList empty_request_parameter)
                    throws RemoteException{
        VOResourceHome home = getInstanceVOResourceHome();
         GetVOListResponse response = new GetVOListResponse();
         response.setValue( home.getVOList() );
         return response;
    }
}
```

Die Aufgabe der Methoden des VOFactoryService sind im Einzelnen:

getInstanceVOResourceHome() Eine private Methode, die eine Referenz auf das VOResourceHome zurück gibt. Der Name desselben wurde in der Datei `deploy-jndi-config.xml` unter dem Namen *InstanceHome* abgelegt und wird nun hier referenziert.

getVOList() Erfragt bei dem VOResourceHome eine Liste der aktiven VOs. Diese Liste wird in eine netztransportfähige Antwort, wie in der WSDL des VOFactoryService definiert, verpackt und zurückgegeben.

createVO() Die Hauptaufgabe eines Factory Services ist das Erzeugen von Virtuellen Organisationen. In dieser Methode wird auf das VOResourceHome zugegriffen, und mit dessen `create()`-Methode eine VO erzeugt, bzw. eine Referenz auf eine bestehende eingeholt. Rückgabe ist in beiden Fällen eine Endpoint Reference, gekapselt in ein passendes Antwort-Objekt.

5.5.2 VOInstanceService

Listing 5.13: VOInstanceService.java (gekürzt)

```
package cojocaru.VO.impl;

public class VOInstanceService {
}
```

Im Gegensatz zum Factory Service, der im vorigen Abschnitt besprochen wurde, sind die Aufgaben des Instance-Services also weitaus umfangreicher. Dennoch – wie im Listing 5.13 gezeigt wird, ist hier die Implementierung der Java Klasse wesentlich kürzer. Sie enthält nämlich keine eigenen Funktionen. Man profitiert hier eindeutig von dem Framework des Globus Toolkit 4. (Nachteil ist, dass die Verknüpfung mit Providern und anderen Elementen auf viele verschiedene Dateien verstreut ist, wie später in 5.6 gezeigt wird.)

Listing 5.14: VOInstanceServicePortType.java (gekürzt)

```
public interface VOInstanceServicePortType extends java.rmi.Remote {
    public SetTerminationTimeResponse setTerminationTime(
        SetTerminationTime setTerminationTimeRequest);
    public QueryResourcePropertiesResponse queryResourceProperties(
        QueryResourceProperties_Element queryResourcePropertiesRequest);
    public SetResourcePropertiesResponse setResourceProperties(
        SetResourceProperties_Element setResourcePropertiesRequest);
    public GetResourcePropertyResponse getResourceProperty(
        javax.xml.namespace.QName getResourcePropertyRequest);
}
```

Betrachtet man die vom Globus Build Service erzeugte Implementation des PortType des Instance Service in Listing 5.14, erkennt man die vermissten Methoden aus dem WSRF zum Zugriff auf die Eigenschaften der Ressource.

Die eigentliche Implementation der WSRF-Methoden musste nicht getätigt werden. Globus stellt hierfür Operation Provider bereit, die die gewünschte Tätigkeit unabhängig von der Vererbungshierarchie unseres Service

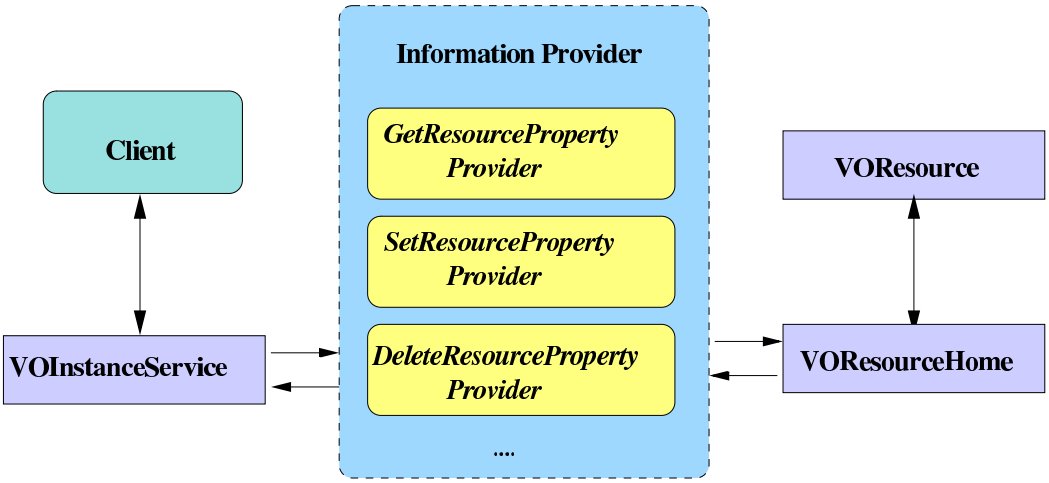


Abbildung 5.8: GT4 Provider

anbieten. Für jede WSRF-Methode, wie in der Abbildung 5.8 dargestellt, bietet die Globus-Umgebung je einen Provider.

Die hier enthaltenen Methoden bewirken im Einzelnen:

getCurrentTime() Gibt die aktuelle Zeit aus Sicht der Ressource zurück. Dies kann verwendet werden, um eine Abstimmung der Lebenszeit einer Ressource bei nicht synchron laufendem Zeitverständnis zu erreichen.

getTerminationTime() Hier wird die Zeit zurückgegeben, zu der die VO aufhören wird zu existieren.

setTerminationTime() Mit dieser Methode kann der VO ein neuer Ablebenszeitpunkt mitgeteilt werden.

setResourceProperties() Diese Methode setzt eine der ResourceProperties. Dabei ist zu beachten, dass nur Kinder des Root-Elementes des Resource Property Dokumentes gesetzt werden können, nicht beliebig tief verschachtelte Informationen. Dies stellt einen schweren Nachteil der Implementation von Virtuellen Organisationen als WSRF-kompatible Ressourcen dar, wie später in Kapitel 6 genauer erklärt wird.

getResourceProperty() Diese Methode gibt eine gewählte Resource Property zurück. Auch hier ist zu beachten, dass nur Root-Elemente abgefragt werden können. Zusätzlich gibt es auch eine Methode, die gleich mehrere ResourceProperties auf einmal abfragt.

queryResourceProperties() Diese lesende Methode bietet die Möglichkeit mit Hilfe von XPath-Anfragen [ClDe 99] auf tiefer verschachtelte Informationen zuzugreifen. Allerdings muss der Aufrufer dann selbst darauf achten, die Antworten wieder in passende Objekte umzuwandeln. Die Antwort enthält nämlich tatsächlich nur die entsprechende XML-Struktur als Ergebnis.

In diesem Abschnitt haben wir uns nur mit den Funktionen des Instance Services auseinandergesetzt. Dabei fehlen uns noch die in seiner WSDL beschriebenen Ressource Eigenschaften, die über den Instance Service manipuliert werden können. Diese werden jetzt im nächsten Abschnitt direkt zusammen mit ihrer Implementierung in der VOResource vorgestellt.

5.5.3 VOResource

Entsprechend den in der WSDL-Beschreibung des VOInstanceServices beschriebenen Eigenschaften müssen auch diese Informationen in der Implementation der VOResource abgelegt werden. Auch hier erhält man wieder Unterstützung vom Globus Build Service. Dieser erzeugt nämlich aus der WSDL Beschreibung Java Klassen für die verschachtelten Elemente der Eigenschaften, wie sie in Abbildung 5.9 dargestellt werden.

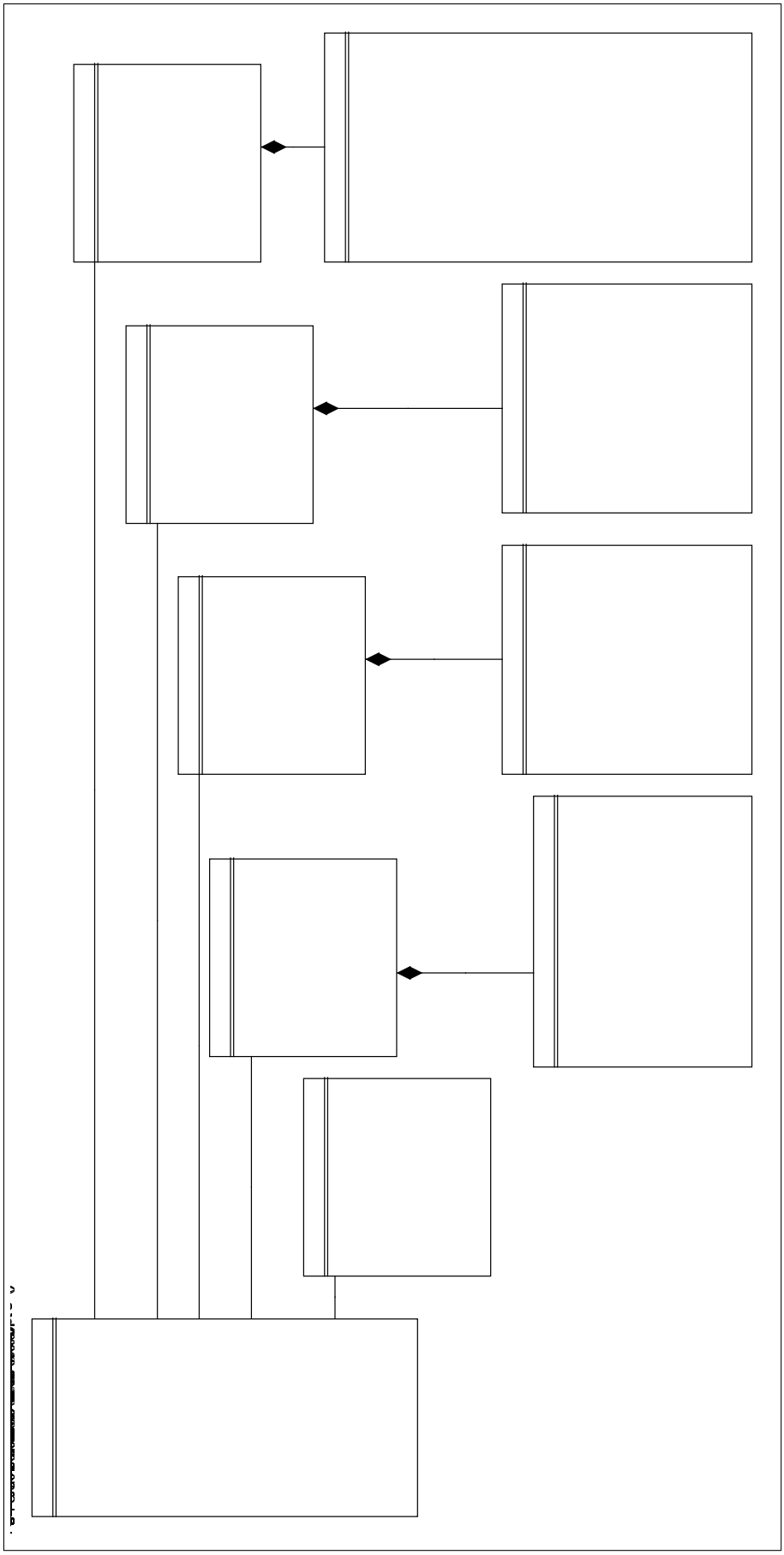


Abbildung 5.9: VO Ressource Properties UML Klassendiagramm

Für die `VOPersons` werden zum Beispiel die Klassen *VOPersons* und *VOPerson* erzeugt. Erstere enthält dabei eine Methode *getVOPerson(): VOPerson[]*, die den Array von Personen zurückgibt. Die *VOPerson*-Klasse enthält Accessormethoden zu den einzelnen Eigenschaften einer Person, die jeweils den Namen der Elemente entsprechen (also getVOPersonName() etc.).

Zusätzlich zu den für den Client direkt sichtbaren Eigenschaften muss die Ressource natürlich auch den bereits erwähnten ResourceKey besitzen. Dieser wird auch vom Interface *ResourceIdentifier* (ResourceID) gefordert. Ein Auszug aus VOResource.java in Listing 5.15 zeigt beispielhaft die Implementierung.

Listing 5.15: VOResource.java (Auszug)

```
public class VOResource implements Resource, ResourceIdentifier,
        ResourceLifetime, ResourceProperties {
    private Object key;
    private ResourcePropertySet propSet;
    private VOPersons vOPersons;
    public Object getID() { return this.key; }
    public VOPersons getVOPersons() { return vOPersons; }
    public void setVOPersons( VOPersons persons ) throws RemoteException {
        this.vOPersons = persons;
    }
    public Object initialize(String name) throws RuntimeException {
        this.propSet =
            new SimpleResourcePropertySet( VOQNames.RESOURCE_PROPERTIES );
        ResourceProperty personsRP =
            new ReflectionResourceProperty(VOQNames.RP_PERSONS,"VOPersons",this);
        this.propSet.add(personsRP);
    }
    this.key = new Integer(hashCode());
    return key;
```

Mehrere Punkte fallen an diesem Ausschnitt auf:

- *Simple ResourcePropertySet*. Diese Methode der Speicherung von Ressource-Eigenschaften ist nicht persistent. Für unsere konzeptionelle Implementierung ist dies ausreichend. Allerdings sollte dies für einen produktiven Einsatz in ein persistentes Modell gewandelt werden, um Informationsverlusten durch Ausfälle und Übertragbarkeit vorzubeugen.
- *ReflectionResourceProperty*. Die einzelnen Resource Properties stehen mit demselben Namen als Klassenvariable, wie sie in der WSDL definiert wurden. Einziger Unterschied ist der von Java geforderte kleine Anfangsbuchstabe. Weiter gibt es Accessormethoden mit get...und set..., über die auf die jeweilige Property zugegriffen werden kann.
- *QName*. Über eine zentrale Registrierungsstelle für die Namen der Properties wird die Korrektheit und Eindeutigkeit sichergestellt.

Listing 5.16: Festlegung des Datentyps für die Objekt ID

```
<parameter>
   <name>resourceKeyType</name>
   <value>java.lang.Integer</value>
</parameter>
```

Als interner Identifikator wird ein nicht näher spezifiziertes Objekt mit Namen „Key“verwendet. Die Methode `getId()` gibt diesen zurück. Der tatsächliche Typ soll per Design nicht betrachtet werden. Er wird in der Datei `deploy-jndi-config.xml` festgelegt, wie in Listing 5.16 gezeigt. Auch der Client soll sich nicht um den Datentyp kümmern, sondern ihn einfach nur weiterreichen.

In unserer Implementierung haben wir für den Identifikator der Ressource einen Integer Wert verwendet, der sich aus dem Hash über das Objekt errechnet. Bei einer großen Anzahl von VOs sollte dies eventuell auf eine andere Methode umgestellt werden. Auch denkbar wäre es, hier gleich den Namen der VO zu verwenden, da wir diesen ja als eindeutig definiert haben.

5.5.4 VOResourceHome

Das *VOResourceHome* übernimmt die Registratur und die Verwaltung der Virtuellen Organisationen, genauer der VOResourcen.

Wir werden ein einziges RessourceHome für alle unsere erstellten Ressourcen verwenden. Darauf greifen auch die beiden oben beschriebenen Services zu, der VOFactoryService, der neue Ressourcen erstellt und deren Keys zu dem Client zurück gibt, sowie der VOInstanceService, der die entsprechenden Ressourcen mit Hilfe eben dieses Keys sucht, um Operationen auf den Ressourcen auszuführen.

Listing 5.17: VOResourceHome.java (gekürzt)

```
public class VOResourceHome extends ResourceHomeImpl {
    public ResourceKey create(String name, String admin, String instanceURI) {
        for ( Iterator i = resources.entrySet().iterator(); i.hasNext(); ) {
            Map.Entry me = (Map.Entry) i.next();
            VOResource vor = (VOResource) me.getValue();
            if ( vor.getVOName().equals(name) ) {
                return (ResourceKey) me.getKey();
            }
        }
        VOResource voResource = (VOResource) createNewInstance();
        voResource.initialize(name);
        ResourceKey key = new SimpleResourceKey(keyTypeName, voResource.getID());
        add(key, voResource);
        /* Hier erzeugen wir eine Gruppe "Administratoren", deren Mitglieder die
         * Gruppe modifizieren koennen.
         * Initiales Mitglied ist der Benutzer, der die VO angelegt hat.*/
        return key;
    }
}
```

Die wichtigste Methode des *VOResourceHome*s ist `create()`. Sie ist in Listing 5.17 dargestellt und sorgt für das Auffinden einer bereits bestehenden, bzw. für die Erstellung und Initialisierung einer neuen Ressourcen-Instanz. In beiden Fällen wird der Key zurückgegeben, über den die Ressource dann angesprochen werden kann.

Ressourcen-Initialisieren bedeutet, dass für die gewünschte Ressource die Resource Properties, wie im vorigen Abschnitt beschrieben, erzeugt werden. Diese sind initial natürlich leer.

Außerdem wird im *VOResourceHome* eine Administratoren-Gruppe angelegt, die Zugriff auf diese VO hat. Der Aufrufer wird in diese neue Gruppe eingetragen. Dies ist notwendig, da sonst die von uns entwickelte Security keinen weiteren Zugriff auf diese VO zulassen würde.

5.5.5 VOMemberInfo

Da die Klasse, die Informationen über den Aufrufer und dessen Rechte beinhaltet, sehr umfangreich ist, wird die Funktionsweise in Listing 5.18 in Pseudocode dargestellt. Es gibt hier zwei Fälle zu unterscheiden:

1. VO wird angesprochen: Wenn eine Methode einer VO aufgerufen wird, hat der PIP an dieser Stelle schon die Information, in welcher VO er nach Rechten suchen muss. Daher werden zur Verwaltung einer VO auch nur Mitglieder der VO mit entsprechenden Rechten zugelassen.
2. Andere WS-Resource wird angesprochen: Zum Überprüfen der Rechte des Aufrufers werden alle VOs durchsucht. Bei den VOs, bei denen er Mitglied ist, werden seine Rechte gesammelt.

Listing 5.18: VOMemberInfo.java (Pseudocode)

```
public class VOMemberInfo implements PIP {
    public void collectAttributes(Subject peerSubject, MessageContext context,
            QName operation) throws AttributeException {
```

```
        sammlePrincipalsDesAufrufers();
        sammleEPRderAufgerufenenRessource();
        resource = holeObjektDerGefordertenRessource();
        if ( resource.instanceof(VOResource.class)) {
            sammleInfoAusVO(resource);
        } else {
            foreach vo in AlleVOs {
                sammleInfoAusVO(vo);
            }
        }
        fuegeZuCredentialsHinzu( this );
    }
    private void sammleInfoAusVO(VOResource vo ) {
        sammleGruppenZuDenenDerAufruferDirektGehoert( vo );
        sammleGruppenZuDenenDerAufruferIndirektGehoert( vo );
        sammleRechteDesAufrufers( vo );
    }
}
```

Die gesammelten Informationen werden dabei in der Klasse global gespeichert. Die Instanz des PIPs wird den Attributen des Benutzers hinzugefügt. Der PDP, der diese Informationen dann über Accessor-Methoden auswertet, wird im nächsten Abschnitt erklärt.

5.5.6 VOMemberPermit

Um diese Entscheidungsfindung durchzuführen, besitzt der PDP, bei uns VOMemberPermit, eine Methode `isPermitted()`, deren Ergebnis vom Globus Security Framework im Rahmen der Authorization Chain abgefragt wird. Einen Auszug unseres PDPs zeigt Listing 5.19.

Listing 5.19: VOMemberPermit.java (Auszug)

```
public boolean isPermitted(Subject peerSubject, MessageContext context,
        QName operation) throws AuthorizationException {
    boolean result = false;
    String operationMethod = operation.toString().substring(
        operation.toString().indexOf("}") + 1
        );
    Set credentials = peerSubject.getPublicCredentials();
    Iterator cred_i = credentials.iterator();
    VOMemberInfo info = null;
    while (cred_i.hasNext()) {
        Object o = cred_i.next();
        if ( o instanceof VOMemberInfo) {
            info = (VOMemberInfo) o;
        }
    }
    if (! (info == null) ) {
        EndpointReferenceType[] objects = info.getObjects();
        EndpointReferenceType calledEPR = info.getCalledEPR();
        for ( int i = 0; i < objects.length; i++ ) {
            EndpointReferenceType obj = objects[i];
            if ( obj.toString().equals(calledEPR.toString())) {
                /* Hier wird noch geprueft, ob in dem Recht eine Methode mit
                   Asterisk angegeben wurde, also z.B. get*, diese Maske
                   wird dann ueberprueft.
                */
                result = true;
                break;
            }
```

```
        }
    }
    return result;
}
```

Hierbei wird die Endpoint Reference, die zum Aufruf vom Client verwendet wurde, mit denjenigen verglichen, auf die er zugreifen darf. Die Entscheidungsfindung ist daher linear über die Anzahl der zugeteilten Rechte.

5.5.7 VOClient

Nachdem wir nun die Hauptkomponenten unserer Implementierung ausführlich beschrieben haben, wäre jetzt angebracht, die Zusammenhänge, die in der Abbildung 5.10 schematisch gezeichnet sind, kurz zu beschreiben. Zu beachten ist, dass hier die Kommunikationsbeziehungen zwischen den Java-Klassen und keine UML-Relationen dargestellt sind. Soll heissen, die Pfeile stellen Aufrufe von Methoden einer anderen Klasse dar. Insgesamt lässt sich so erkennen, auf welchem Weg die Anfrage des Clients letztendlich zur VOResource gelangt.

Auf der Client-Seite befinden sich die Klassen *VOClient, VOFactoryServicePortType* und *VOInstanceServicePortType*. Alle anderen relevanten Java-Klassen befinden sich auf dem Server. Der Client kann eine VO nur mit Hilfe vom VOFactoryService erzeugen. Dafür benötigt er die eindeutige URI dieses Web Services.

Um die `create()`-Methode auszuführen, wird der VOFactoryServicePortType benötigt. Dieser stellt für den Client einen Stub bereit, auf dem die Methode wie bei einem normalen Objekt aufgerufen werden kann. Der Stub kapselt die Anfrage dann gemäß der WSDL-Beschreibung des Web Services in eine SOAP-Nachricht, und schickt sie zum Grid.

Ein beispielhafter Ablauf, der auch die folgenden Schritte auf der Client Seite aufzeigt, ist in Listing 5.20 gezeigt.

Listing 5.20: VOClient.java (Ausschnitt: VO Erzeugung)

```
public static void main( java.lang.String[] args ) {
    String factoryURI = args[0];
    EndpointReferenceType factoryEPR = new EndpointReferenceType();
    factoryEPR.setAddress(new Address(factoryURI));
    VOFactoryServiceAddressingLocator factoryLocatorVO =
        new VOFactoryServiceAddressingLocator();
    VOFactoryServicePortType voFactory =
        factoryLocatorVO.getVOFactoryServicePortTypePort(factoryEPR);
    //Jetzt koennen wir die Methoden der VOFactoryService auf voFactory
      ausfuehren

    EndpointReferenceType instanceEPR = null;
    VOInstanceServicePortType voInstance = null;
    instanceEPR = voFactory.createVO(voName).getEndpointReference();
    VOInstanceServiceAddressingLocator instanceLocatorVO =
        new VOInstanceServiceAddressingLocator();
    VOInstanceServicePortType voInstance =
        instanceLocatorVO.getVOInstanceServicePortTypePort( instanceEPR );
    //Jetzt koennen wir die Methoden des VOInstanceService auf voInstance
      ausfuehren

    GetResourcePropertyResponse personenRP =
        voInstance.getResourceProperty(VOQNames.RP_PERSONS);
    // etc.
}
```

Anhand der vom Client spezifizierte Service-URI wird der Locator [Glob 05] (GT4 Komponente) den benötigten PortType lokalisieren. Der PortType wandelt die entgangene Information in einer SOAP-Nachricht, die

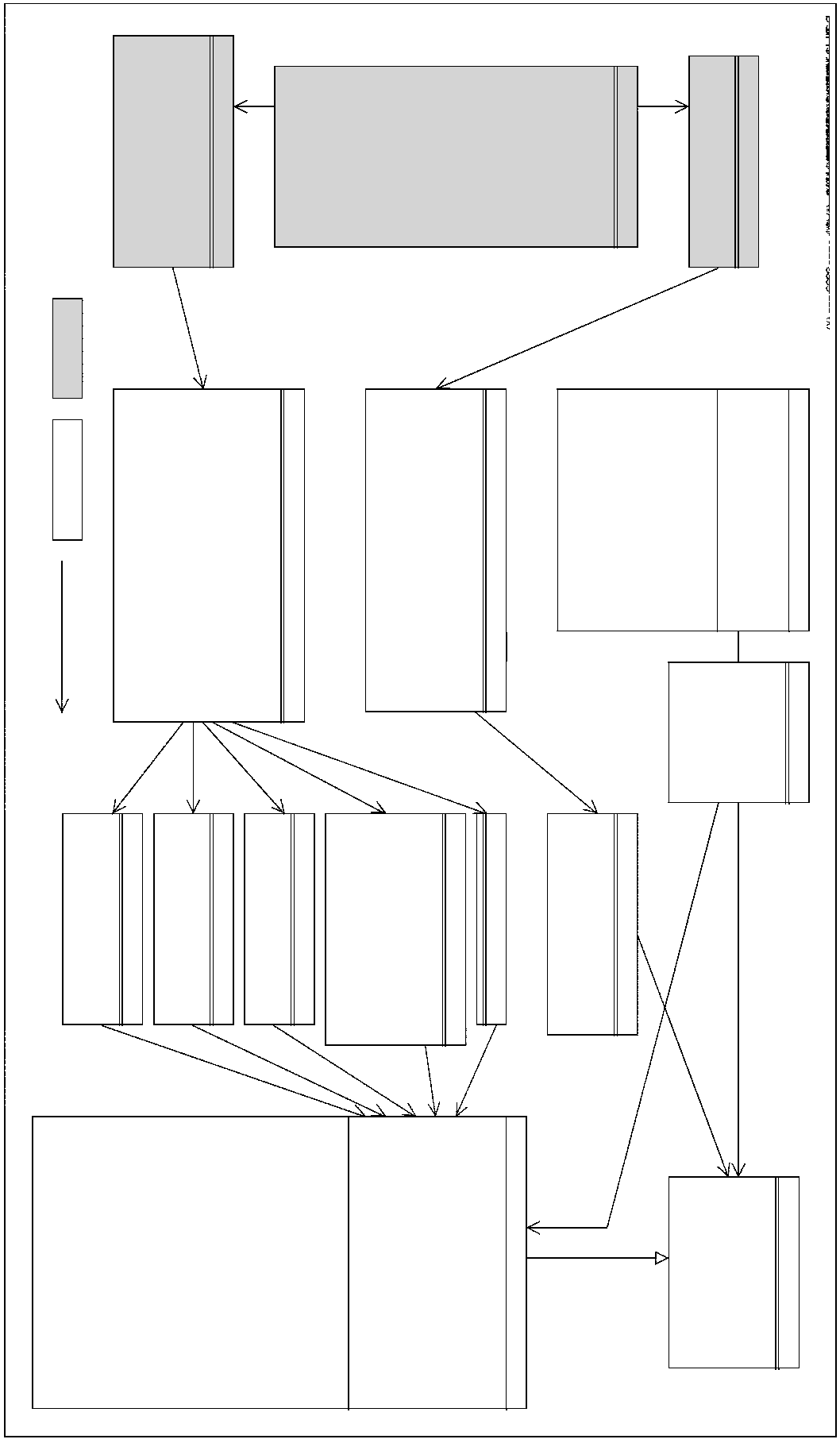


Abbildung 5.10: Kommunikationsbeziehungen einer VO

auf der Server-Seite vom VOFactoryServicePortTypeSOAPBindingStub wieder in einen Java-Aufruf an den VOFactoryService übersetzt wird.

Mit Hilfe vom Security Descriptor des Dienstes wird festgestellt, ob der Benutzer sich gegenüber Grid authentisieren kann bzw. zur Bildung einer neuen VO berechtigt ist. Die VORessourceHome legt eine neue VO-Ressource an, die durch leere Properties, die in der VOResource gespeichert sind, initialisiert wird. Die RessourceID und die URI des VOInstanceServices wird an den Client zurückgeschickt. Ab diesem Moment darf der Client weitere Operationen auf die Ressource ausführen.

5.6 Build und Deployment

Nachdem wir nun die einzelnen, für das Thema dieser Arbeit entwickelten Komponenten beschrieben haben, betrachten wir nun deren Auslieferung und Inbetriebnahme.

Noch zu erwähnen ist, dass die Steuerdateien, Java-Klassen und die WSDL-Beschreibungen, in einer strikt definierte Baum-Struktur angelegt werden müssen. Wird die Struktur nicht eingehalten, startet der Container nicht. Wie die Dateien für diese Arbeit hierarchisch angelegt sind, ist in der Abbildung 5.11 zu betrachten.

Wie bereits erwähnt, ist das Globus Toolkit 4 [Soto 05] die Basis unserer Implementierung [7]. Diese Arbeit baisert auf Version GT 4.0.4, ab April 2007 steht die GT Version 4.0.5 jedem zur Verfügung. Um mit Hilfe des GT4 entwickeln zu können, sind weitere Softwarekomponenten notwendig. Diese und teilweise deren Einrichtung werden im Folgenden beschrieben.

Nicht auf allen Systemen ist die nachträgliche Installation aller dieser Komponenten erforderlich. Einige davon gehören zur Grundinstallation von Linux Distributionen. Globus lässt sich auch unter Windows oder anderen Unices betreiben, allerdings wurde die Entwicklung im Rahmen dieser Diplomarbeit unter Linux durchgeführt. Aufgrund der verwendeten Programmiersprache Java sollte es allerdings ohne Probleme möglich sein, unsere Ergebnisse auf anderen von Globus unterstützten Betriebssystemen zum Einsatz zu bringen.

Auf dem zur Entwicklung verwendeten Rechner ist beispielsweise die Version Java(TM) SE Runtime Environment (build 1.6.0_01-b06) im Einsatz. Die Verwendung des Software Development Kits (SDK) ist notwendig, da die erstellten Quelldateien in Bytecode umgewandelt werden müssen. Der hierzu erforderliche Compiler ist nur in dem umfangreicheren SDK enthalten [8].

Die erforderliche Software laut GT4-Installations-Dokumentation lässt sich in zwei Teile aufteilen. Der eine Teil ist nur notwendig, falls Globus aus Quellen übersetzt werden soll (siehe Anhang C). Der andere Teil ist für die Übersetzung der hier erstellten Web Services und Klassen notwendig. Zu den erforderlichen Komponenten gehören:

Apache Axis

Apache eXtensible Interaction System (Axis) [The 06] ist eine Bibliothek der Apache Software Stiftung. Sie bietet die benötigten Funktionen zur Kapselung der über das Netzwerk gesendeten Nachrichten in SOAP/XML-Format an. Diese Kommunikationsmethode wird von Web Services bevorzugt, und daher auch von den hier Entwickelten verwendet. Axis ermöglicht den Web Services und ihren Clients, eine standardisierte und definiert strukturierte Kommunikation aufzubauen. Axis ist in Java und C++ implementiert, setzt aber nur den SOAP Standard um. Daher können natürlich auch andere entsprechende Bibliotheken zur Verwendung mit einem Axis-basierenden Web Service herangezogen werden.

Apache Axis ist eine Neuentwicklung und Nachfolger von Apache SOAP, das auf dem IBM-Framework IBM SOAP [Cohe 01] basierte. Ziel dieser Neuentwicklung war es eine höhere Geschwindigkeit, Flexibilität, Komponentenorientierung, Abstraktion des Transportframeworks, sowie die Unterstützung von WSDL zu erreichen.

[7] Zum Download unter http://www.globus.org/toolkit/downloads/4.0.4/
[8] JAVA J2SE 1.4.2+ SDK von Sun notwendig

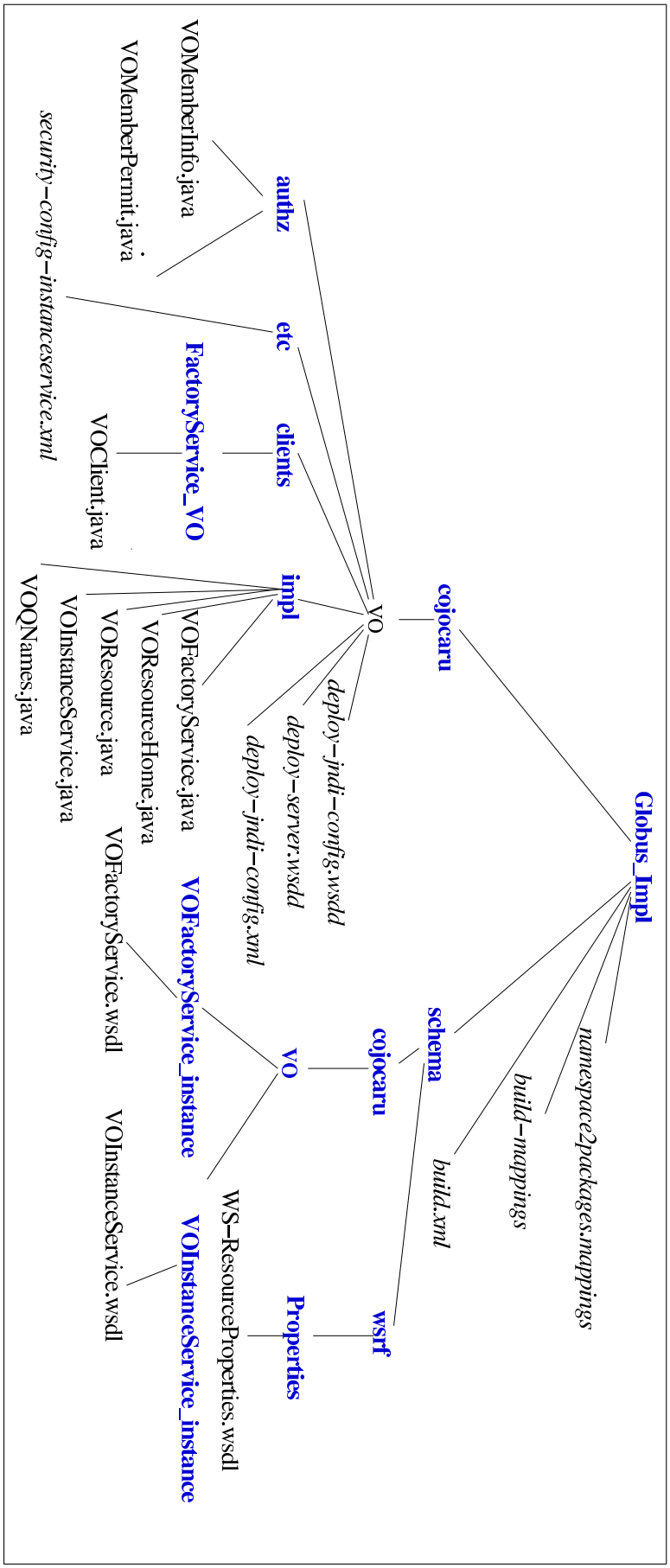


Abbildung 5.11: Verzeichnisstruktur vom Globus Build Service (Beispiel)

Die höhere Geschwindigkeit erreicht die Axis Library durch Verwendung des Simple API for XML (SAX)-Parsers [Brow 02] im Gegensatz zu Apache SOAP, das noch auf einem langsameren Document Object Model (DOM)-Parser [ABC+ 98] aufbaute.

Um die entwickelte Web Services an Axis anzubinden, wird zusätzlich der sogenannte Web Service Deployment Descriptor (WSDD) verwendet, zum Deployment und Undeployment von den Services. Die Datei trägt den Namen `deploy-server.wsdd` 5.6 für den serverseitigen Service. In ihr stehen Informationen über die zu startenden Services und ihre Interfacebeschreibungen und Parameter.

Für diese Arbeit wird die Apache Axis Version 1.4 verwendet, die seit April 2006 jedem frei zur Verfügung steht [9].

Ant

Das in Java implementierte OpenSource Werkzeug *Ant* steht für „Another Neat Tool“ [BDH+ 06] und ist ein in Java entwickelte Tool, das die Möglichkeit bietet, aus Quellcode Programme zu erzeugen. Ant ist aber selbst keine Programmiersprache. Das Werkzeug ist „universell“ anzuwenden und ist mit *Make*-Tool (einem UNIX-Programm, dass in Abhängigkeit von vorher definierten Bediengungen entsprechende Befehle ausführt) vergleichbar. Mit dem Befehl `ant -version` wird die Version vom Ant [10] -Tool überprüft.

Ant ist ein Werkzeug zur automatisierten Codeübersetzung und wird bei Globus indirekt zum Vorbereiten der Web Services und abhängiger Klassen verwendet.

Was und wie das plattformunabhängige Tool ausführen und kompilieren soll, wird in der `build`-Datei, einer XML-Beschreibung, vordefiniert. Die Hauptfunktion von Ant in Bezug auf Globus besteht darin, eine neue GAR-Datei zu erstellen, in der alle kompilierte Java-Klassen der Web Services abgelegt werden. Das Erzeugen einer GAR-Datei ist kein einfacher Vorgang und beinhaltet mehrere Schritte, die von Ant übernommen werden, wie [BDH+ 06]:

- Erzeugung von Stub-Klassen aus der WSDL-Beschreibung
- Kompilieren von Stub-Klassen
- Kompilieren von Service Implementation
- Ablegen von Dateien in einen spezifischen GAR-Verzeichnis etc.

Für die Erstellung einer GAR-Datei benötigt *Ant* mehrere Informationen aus unterschiedlichen Dateien. Die Abbildung 5.12 zeigt uns die benötigte Komponenten beim Erzeugung vom GAR-file: die WSDL-Beschreibung und die Java-Implementierung von den Services, den Deployment Descriptor, build-Datei und die GT4 build files (da wir die GT4 Testumgebung benutzen).

In der `build`-Datei, wie oben schon erwähnt, stehen Ant nützliche Informationen bezüglich des Kompilierens zur Verfügung, also was, wie und in welchem Verzeichnis kompiliert wird.

Das `globus-build-service-Skript` ist vom Globus Service Build Tools (GSBT)-Projekt [11] entwickelt worden. Das Kompilieren von den Web Services, die in der `build.mapping` 5.21-Datei eingetragen sind, kann durch das Kommando `./globus-build-service.sh` gestartet werden.

Listing 5.21: „build.mapping-Einträge (Beispiel) “

```
VO,cojocaru/VO,
schema/cojocaru/VO/VOInstanceService_instance/VOInstanceService.wsdl,
schema/cojocaru/VO/VOFactoryService_instance/VOFactoryService.wsdl
```

Bei der Befehlseingabe werden nicht nur die WS-Klasse (Java) kompiliert, also `.class`-Dateien erzeugt, sondern auch die WSDL-Dateien um z.B. WSRF-Methoden, die zu importieren sind, expandiert. Aus den in WSDL definierten ResourceProperties werden pro Property je eine Java-Klasse automatisch erzeugt und

[9] Zum Download unter http://ws.apache.org/axis/
[10] Falls Java 1.5 im Einsatz, Ant 1.5.1+; 1.6.1+ notwendig
[11] Weitere Informationen unter http://gsbt.sourceforge.net/

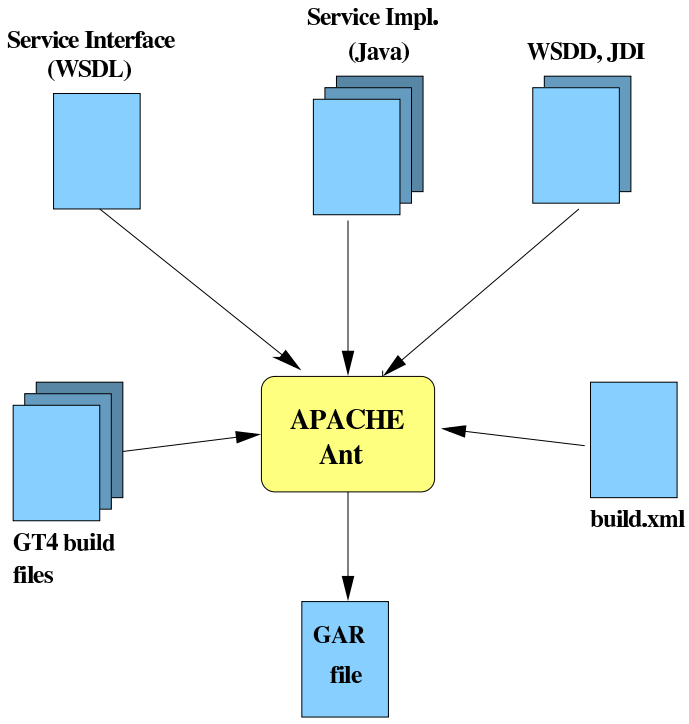


Abbildung 5.12: Erzeugung eines GAR-files mit Ant [SoCh 06]

kompiliert (z.B. VOPersons, VOGroups etc.). Schliesslich werden die Services und dazugehörige Java-Klassen in eine `GAR`-Datei verpackt (hier `cojocaru_VO.gar`), die später, während des `deploy`-Prozesses für den Globus-Container verwendet wird 5.6.

Für diese Arbeit wird die aktuelle Ant Version 1.7 verwendet, die seit Dezember 2006 jedem frei zur Verfügung steht[12]. In den nächsten Abschnitten werden zuerst die notwendige Steuerdateien dargestellt, um dann am Ende das Verfügbarmachen der Services und der Sicherheitskomponenten zu erklären.

5.6.1 Web Services Deployment Descriptor (WSDD)

Die entwickelten Web Services müssen innerhalb eines Servlet-Containers gestartet werden, der die Anfragen vom Netzwerk entgegennimmt und an den entsprechenden Dienst weiterleitet. Wir haben dafür den Globus-eigenen Container [SoCh 06] genutzt, es ist aber auch möglich, den verbreiteten Servlet-Container Tomcat [13] dafür zu verwenden.

Der Prozess zum Bekanntmachen der Web Services im Container wird als *Deployment* oder *Softwareverteilung* bezeichnet. Der *Deployment Descriptor* ist eine XML-Beschreibung, die Metainformationen aller Servlets definiert.

In unserem Fall beinhaltet der Descriptor die Information über die beiden Dienste, *VOFactoryService* und *VOInstanceService*, die wir für das Erstellen der `VORessource` und den Zugriff auf die VOResource benutzen. Mit Hilfe vom Descriptor wird der Zugriff vom Client auf den Web Service ermöglicht. Das geschieht durch den sogenannten *Web Service Container*, der nicht anders als ein Oberbegriff für die Application-, HTTP-Server und SOAP-Engine ist.

Ein Beispiel für eine WSDD-Datei, die Informationen über unsere Services beinhaltet, ist im Anhang D zu finden. Im Folge werden wir die wichtigsten Descriptor-Elemente des VOInstanceServices auflisten. Das Element „Service name“ 5.22 zeigt beispielsweise wo unser VOInstanceService abgelegt wurde. Die wichtigsten

[12] Zum Download unter http://ant.apache.org/bindownload.cgi
[13] Weitere Informationen unter http://tomcat.apache.org/

Listing 5.22: deploy-server.wsdd (gekürzt)

```
<deployment name="defaultServerConfig">
    <!-- Instance service -->
    <service name="cojocaru/VO/VOInstanceService" provider="Handler" use="literal" style="document">
        <parameter name="className" value="cojocaru.VO.impl.VOInstanceService"/>
        <wsdlFile>
            share/schema/cojocaru/VO/VOInstanceService_instance/VOInstanceService_service.wsdl
        </wsdlFile>
        <parameter name="allowedMethods" value="*"/>
        <parameter name="handlerClass" value="org.globus.axis.providers.RPCProvider"/>
        <parameter name="scope" value="Application"/>
        <parameter name="providers" value="GetRPProvider GetMRPProvider SetRPProvider QueryRPProvider"/>
        <parameter name="loadOnStartup" value="true"/>
        <parameter name="securityDescriptor" value="etc/cojocaru_VO/security-config-instanceservice.xml"/>
    </service>
    <!-- Factory service -->
    <service name="cojocaru/VO/VOFactoryService" provider="Handler" use="literal" style="document">
        <parameter name="className" value="cojocaru.VO.impl.VOFactoryService"/>
        <wsdlFile>
            share/schema/cojocaru/VO/VOFactoryService_instance/VOFactoryService_service.wsdl
        </wsdlFile>
        <parameter name="allowedMethods" value="*"/>
        <parameter name="handlerClass" value="org.globus.axis.providers.RPCProvider"/>
        <parameter name="scope" value="Application"/>
        <parameter name="instance" value="cojocaru/VO/VOInstanceService"/>
    <parameter name="loadOnStartup" value="true"/>
    </service>
</deployment>
```

Parameter bewirken im einzelnen:

className Der Parameter „className“ spezifiziert die Klasse VOInstanceService, die die Schnittstelle zu dem Instance Service implementiert. Hierdurch erkennt der Container, an welches Java-Objekt er die Anfrage weiterleiten muss.

wsdlFile Der Parameter „wsdlFile“ teilt dem Web Services Container mit, wo die WSDL-Beschreibung vom Instance Service zu finden ist.

loadOnStartup Mit Hilfe des Parameter „Load on Startup“ kann spezifiziert werden, ob der Instance Service beim Starten vom Container gleich gestartet wird oder nicht.

securityDescriptor Der Security Descriptor konfiguriert die Authorization Chain, die bereits weiter oben erwähnt wurde (Siehe 5.5.6 und 5.5.5).

instance Der Factory Service erhält zusätzlich noch die Information, unter welcher Service-URI der Instance Service für die von ihm erzeugten Objekte zu finden ist.

Die weiteren Parameter, wie „allowedMethods“, „handlerClass“ und „scope“ sind in jedem Web Service Code vorhanden. Diese heissen auch „Common Parameters“.

5.6.2 Java Naming and Directory Interface (JNDI)

JNDI [Lee 02] ist ein Interface, für Namens- und Verzeichnisdienste, innerhalb der Java-Programmiersprache. Das Ziel von JNDI ist die Zugriffsmöglichkeiten auf komplexe Informationen mit Hilfe von einfachen Namen zu bieten. Das Interface ist implementierungsunabhängig und wird meist dazu benutzt, in Java Platform, Enterprise Edition (J2EE) [JBC+ 06]-Anwendungen Objekte in einem Netzsegment zu registrieren, die weiterhin mit Java-Programmen (durch ein directory) remote aufgerufen werden können.

Seit der Version 1.4, die auch für unsere Implementierung verwendet wurde, gehört JNDI zum J2EE. Für die älteren Java-Versionen werden Erweiterungen benötigt, die bei der offiziellen Sun-Seite [14] zum Download bereitstehen. Fast alle Verzeichnis-und Namensdienst-Arten werden von JNDI unterstützt, wie Common Object Request Broker Architecture (CORBA) [Wall 07]-, Dateisystem, Lightweight Directory Access Protocol (LDAP) [Open 98] etc.

Ein *Namensdienst* liefert bei der Angabe eines Namens genau ein oder kein Objekt. Als Beispiel wäre der Domain Name Service (DNS)-Server, der zu jedem PC die IP-Adresse zurückgibt. Der Dienst kann nicht nur ein Objekt, sondern auch eine Referenz (URI-Addresse) darauf zurückliefern.

Der *Verzeichnisdienst* ist hierarchisch strukturiert, wie z.B. das UNIX-Dateisystem, das eine Baumstruktur darstellt. In der JNDI-Schnittstelle wird also für die beiden Services festgelegt, welcher Service welches Resource Home verwendet bzw. sucht für jeden Web Service das dazugehörige Resource Home. In unserem Fall verwenden die Factory/ Instance-Services dasselbe *VOResourceHome*.

Im Anhang H ist beispielsweise unser JNDI Deployment File zu finden. Im Weiteren werden wir kurz die Hauptelemente, die auch im unteren Listing 5.23 zu betrachten sind, kurz erklären.

Listing 5.23: deploy-jndi-config.xml (Ausschnitt)

```
<!-- Instance service -->
<service name="cojocaru/VO/VOInstanceService">
    <resource name="home" type="cojocaru.VO.impl.VOResourceHome">
        <resourceParams>
            <parameter>
                <name>resourceClass</name>
                 <value>cojocaru.VO.impl.VOResource</value>
            </parameter>
            <parameter>
                <name>resourceKeyType</name>
                <value>java.lang.Integer</value>
```

[14] Weitere Informationen unter www.sun.com

```
            </parameter>
            <parameter>
                <name>resourceKeyName</name>
                <value>{http://www.cojocaru.de/Namespaces/VO/
                        VOInstanceService_instance}VOResourceKey
                </value>
            </parameter>
        </resourceParams>
    </resource>
</service>
<!-- Factory service -->
<service name="cojocaru/VO/VOFactoryService">
    <resourceLink name="instanceHome"
        target="java:comp/env/services/cojocaru/VO/VOInstanceService/home"/>
</service>
```

Das root-Element „jndiConfig“ beinhaltet zwei Elemente, da wir zwei Web Services entwickelt haben. Die Elemente haben jeweils je ein Attribut „name“, das gleich dem in der WSDD-Datei spezifizierten Service-Name sein muss. Dem Element „service“ wird ein „resource “-Element zum Spezifizieren vom Resource Home des Web Services untergeordnet. Jeder Service beinhaltet ein oder mehrere „resource“-Parameter, wie: `resourceClass` (spezifiziert die Java-Klasse unserer Ressource), `resourceKeyType` (entspricht dem Typ des Keys, das während der Implementierung als Identifikator für das Ressourcen-Objekt erzeugt wurde. In unserem Fall wird ein Hash-Code verwendet, der vom Typ java.lang. Integer ist), `resourceKeyName` (ein Parameter, das ein QName, das in demselben Namespace wie der Service liegt, ist) und weitere.

Die Rolle von JNDI ist sehr offensichtlich: ohne ein vorhandenes Interface wird beim Zugriff auf einem Verzeichnisdienst dessen spezielle Application Programming Interface (API) benötigt. JNDI bietet eine einheitliche Schnittstelle für alle Verzeichnisdienste an.

5.6.3 Namespace mappings

In der Mapping Datei, die die Namespaces auf die Java packages abbildet, muss nun die Information bezüglich des VOFactoryServices und der automatisch generierten Bindings und Service-Schnittstelle eingetragen werden. Dies wird in Listing 5.24 gezeigt.

Listing 5.24: namespace2package.mappings (Auszug)

```
http\://www.cojocaru.de/Namespaces/VO/VOFactoryService
     =cojocaru.VO.stubs.VOFactoryService
http\://www.cojocaru.de/Namespaces/VO/VOFactoryService/bindings
     =cojocaru.VO.stubs.VOFactoryService.bindings
http\://www.cojocaru.de/Namespaces/VO/VOFactoryService/service
     =cojocaru.VO.stubs.VOFactoryService.service
```

Weiter muss noch die Abbildung der Namespaces, die vom VOInstanceService verwendet werden, auf die Java Packages in der Datei namespace2package.mappings eingetragen werden, wie in Listing 5.25 aufgezeigt.

Listing 5.25: namespace2package.mappings (Auszug)

```
http\://www.cojocaru.de/Namespaces/VO/VOInstanceService\_instance
    =cojocaru.VO.stubs.VOInstanceService\_instance
http\://www.cojocaru.de/Namespaces/VO/VOInstanceService\_instance/bindings
    =cojocaru.VO.stubs.VOInstanceService\_instance.bindings
http\://www.cojocaru.de/Namespaces/VO/VOInstanceService\_instance/service
    =cojocaru.VO.stubs.VOInstanceService\_instance.service
```

5.6.4 Übersetzung und Deployment

Bevor der Container mit den Web Services gestartet wird, müssen die ensprechende Services erstmal kompiliert E und deployed F werden.

Den Übersetzungsvorgang übernimmt hierbei der Globus Build Service, ein Service in Form eines Skriptes `globus-build-service.sh` festgelegt. Es verlangt eine spezielle Verzeichniss-Struktur und eine Konfigurationsdatei, die `build.mappings`, auch im Listing 5.21 abgebildet wird. Im selben Abschnitt 5.6 wurde auch der Vorgang des Übersetzens besprochen.

Am Ende dieses Vorgangs entsteht eine `Gar`-Datei, die ein Archiv mit (fast) allen vom Container benötigten Dateien darstellt. Zusätzlich müssen noch, weil nicht von dem Skript erfasst, der PIP 5.5.5 und PDP 5.5.6 von Hand kompiliert werden. Diese können dann in eine separate JAR-Datei gepackt werden.

Der Administrator des Grids erhält nun die `cojocaru_VO.gar` und `cojocaru_VO_authz.jar`. Für erstere verwendet er den Befehl `globus-deploy-gar`, um das Archiv auszupacken und die enthaltenen Dateien an die passende Stelle zu kopieren. Die gepackten Elemente für die Autorisierungskette müssen manuell in das Bibliotheksverzeichnis von Globus verbracht werden.

Beim erfolgreichen Starten des Containers mittels `globus-start-container` werden alle laufende Dienste aufgelistet, wie im Anhang G. Zwischen diesen Services sind auch unsere angelegte VOFactory- und InstanceService zu finden (siehe Listing 5.26).

Listing 5.26: „Ausgabe des WS-Containers (Abschnitt)“

```
[41]: http://192.168.218.21:8080/wsrf/services/TriggerServiceEntry
[42]: http://192.168.218.21:8080/wsrf/services/Version
[43]: http://192.168.218.21:8080/wsrf/services/WidgetNotificationService
[44]: http://192.168.218.21:8080/wsrf/services/WidgetService
[45]: http://192.168.218.21:8080/wsrf/services/cojocaru/VO/VOFactoryService
[46]: http://192.168.218.21:8080/wsrf/services/cojocaru/VO/VOInstanceService
[47]: http://192.168.218.21:8080/wsrf/services/gsi/AuthenticationService
```

Vom diesem Moment an kann der Client gewünschte Anfragen an den entsprechenden Services stellen, z. B. an dem Factory Service bzgl. Erzeugung einer VO. Im nächsten Kapitel werden wir diese Ergebnisse mit den ursprünglich gestellten Anforderungen vergleichen und bewerten.

6 Bewertung der Lösung

Obwohl Grid-Computing ein ziemlich neues Gebiet ist, hat sich in den letzten Jahren rasant entwickelt. In einem Grid bietet sich die Möglichkeit, Grid-Komponenten, wie Ressourcen und Personen, auf Virtuelle Organisationen abzubilden. Da die VOs wichtige Konstrukte sind, war die Idee, sie als WSRF-konforme Ressourcen zu realisieren. In dieser Diplomarbeit haben wir eine Implementierung eines VO-Modells entwickelt, die von uns im Abschnitt 3 definierten Anforderungen erfüllt. Im Verlauf der Entwicklung wurden jedoch einige Mängel an den verwendeten Tools und Definitionen offensichtlich.

In diesem Kapitel wollen wir nun die erreichten Ziele darstellen, sowie einen Anreiz zu weiteren Entwicklungen geben, die die Implementierung weiter perfektionieren könnten.

6.1 Erreichte Ziele

Unsere Arbeit zeigt, dass das Management einer als WS-Resource implementierten VO möglich ist. Die einzelnen Zielaspekte werden in diesem Abschnitt behandelt.

6.1.1 Management einer VO als WS-Resource

Die Implementation folgt der WSRF-Beschreibung einer WS-Resource. Dadurch kann die Verwaltung der Virtuellen Organisation mittels Tools erfolgen, die des Managements von WSRF-konformen Ressourcen mächtig sind.

Wie in Kapitel 5 beschrieben, implementieren wir dabei nicht nur die als zwingend vorausgesetzten Operationen und Eigenschaften des WSRF. Wir haben auch optionale Teile des Rahmenwerks umgesetzt, wie die WSRF-Lifetime. Auch die Methode `setResourceProperty()`, die es ermöglicht, schreibend auf die Eigenschaften einer Ressource zuzugreifen, und damit Änderungen vorzunehmen, wurde eingebaut.

6.1.2 Management des Lebenszyklus einer VO

Unsere Umsetzung ermöglicht die Verwaltung des Lebenszyklus einer Virtuellen Organisation. Für die Erzeugung haben wir hierbei, den Empfehlungen des Globus Toolkits 4 folgend, die Fabrik-Methode angewendet. Die Methode `createVO()` des `VOFactoryServices` gibt dem Aufrufer eine Endpoint Reference auf die neu erzeugte VO zurück. Dabei wird im Hintergrund auch eine erste Initialisierung vorgenommen, es wird eine Administratoren-Gruppe angelegt, deren initiales Mitglied der Aufrufer ist.

Der Betrieb der VO findet mit Hilfe des `VOInstanceServices` statt. Hierüber können alle Eigenschaften der VO bearbeitet werden. Die Schnittstelle wird dabei, wie im vorigen Abschnitt beesprochen, vom WSRF vorgegeben. Die einzelnen WSRF-Funktionen wurden im Instanzdienst nicht direkt implementiert, sondern es wurden die von GT4 bereitgestellten *Provider* 5.4.2 verwendet. Durch die Verwendung dieser vorgegebenen Elemente wurde die Standardkonformität sichergestellt und die Zahl der potentiellen Implementierungsfehler verringert.

Die Auflösung der VO wird über im WSRF optional enthaltenen (geplante oder sofortige) Auflösung gesteuert. Dabei bietet der `VOInstanceService` die Methode `destroy()` an. Diese bewirkt die sofortige Auflösung der VO, und damit aller in ihr definierten Rollen, Personen und Gruppen. Die Zuordnung zu WS-Resourcen wird zwar aufgehoben, aber nicht zerstört. Mittels der Methode `setTerminationTime()`

können wir auch eine zeitgesteuerte Auflösung planen. Diese unterstreicht den temporär begrenzten Charakter einer VO.

6.1.3 Management einer VO als managed object

Im Rahmen des VO Managements ist es nun möglich, die VO als managed object zu betrachten und zu verwalten. Dabei kann ein beliebiges Tool verwendet werden, das den WSRF-Standard unterstüzt.

Die VO unterstützt Abfragen und kann über eine bekannte Schnittstelle administriert werden. Fehler werden unter Verwendung der WS-Basefaults [LiMe 06] weitergegeben. Eine mögliche Optimierung wäre hier das Einsetzen von WS-Notifications [GHM 06]. Dadurch kann sich ein Managementsystem über Veränderungen in der Konfiguration einer VO informieren lassen, ohne diese ständig abfragen zu müssen.

Im nächsten Abschnitt wollen wir andere Entwicklungsmöglichkeiten besprechen, die das Resultat dieser Arbeit weiterführen und noch mehr optimieren können.

6.2 Mögliche weitere Entwicklungen

Bei der Implementierung der zu Beginn dieser Arbeit entwickelten Anforderungen mussten wir teilweise Kompromisse bezüglich Komplexität und Aufwand eingehen. Wir haben uns auf einen gemeinsamen Nenner beschränkt, der die Anforderungen erfüllt und den Rahmen dieser Diplomarbeit so wenig wie möglich überschreitet.

In diesem Abschnitt werden wir die Stellen aufzeigen, an denen Weiterentwicklungsmöglichkeiten und Potential für Verbesserungen besteht. Wichtig dabei ist, im Gedächnis zu behalten, dass der implementierte Ansatz *as is* funktioniert.

6.2.1 Erweiterung zur WS-ServiceGroup

In dieser Arbeit wurde die Aufnahme von WS-Resourcen in eine VO über die Speicherung der Endpoint Reference implementiert. Im WSRF wird auch eine „ServiceGroup“ spezifiziert.

Über eine ServiceGroup wird, laut Beschreibung in [MSB 06], den Zugriff auf die enthaltenen WS-Resourcen ermöglicht. Dabei ist es auch möglich, Zugriffsbeschränkungen zu implementieren. Allerdings besteht die ServiceGroup nicht auf exklusiver Zugehörigkeit. Entsprechend müssten die Definitionen angepasst werden.

Die Implementation als ServiceGroup würde das implementierte Verfahren bei der Berechtigungs-Durchsuche aller existierenden VOs vereinfachen. Da laut Standard über die ServiceGroup auf einen Web Service zugegriffen wird, könnte hier eine konkrete Performance- und Designverbesserung erreicht werden.

6.2.2 Erweiterung des Factory Services um einen Zustand

In unserer Definition besitzt der Factory Service keinen Zustand. Es gibt daher prinzipiell keine Möglichkeit, Informationen über bestehende VOs zu erhalten. In unserer Implementation haben wir eine zusätzliche Funktion eingebaut, die die Liste der VOs zurückgibt.

Eine Weiterentwicklungsmöglichkeit an dieser Stelle besteht darin, zusätzlich dem Factory Service das Interface einer zustandsbehafteten WS-Resource zu geben. Dabei könnte auch die Liste der VOs als eine Resource Property behandelt werden. Die „Ressource“ des Dienstes wäre dann das Resource Home, in dem die „Information“ über die erzeugten VOs gespeichert ist. So könnte man mit Hilfe einer `getResourceProperty()`-Methode die Liste der gewünschten VOs aufrufen. Auch könnte man die Liste der VO-Namen gleich mit den entsprechenden EPRs paaren, so dass ein weiterer Aufruf zum Abholen desselben nicht mehr notwendig wäre.

Ein Schreibzugriff mittels `setResourceProperty()` macht an dieser Stelle allerdings kaum Sinn, da sonst das Factory-Pattern ausgehebelt würde.

6.2.3 Ausgliedern der Personenbezogenen Informationen

Die personenbezogenen Informationen, wie Kontaktdaten, organisatorische Zugehörigkeit etc. werden in unserer Implementation direkt in der VO gespeichert.

Für den Fall, dass eine Person zu mehreren VOs gleichzeitig gehört, entsteht hier eine mehrfache Datenhaltung. Zwar wäre es durchaus denkbar, dass eine Person für verschiedene VOs unterschiedliche Kontaktinformationen hinterlegen will, ist es jedoch generell anzunehmen, dass einer Person auch genau ein Datenblatt zugeordnet werden kann. Weiterhin existiert sicherlich bereits vom Grid selbst eine Nutzerverwaltung.

Wir schlagen daher vor, die Nutzerinformation an einen zentralisierten Dienst zu übertragen, an einen Verzeichnisdienst zum Beispiel. Dieser enthält für jede Person eindeutige Identifikatoren, die dann referenziert werden können. Dadurch kann sich ein VO-Verwalter ausschließlich auf das VO Management konzentrieren, ohne sich um die Nutzerverwaltung kümmern zu müssen.

6.2.4 Erweiterung des WSRF um adressierte Manipulation

Der Einsatz vom WSRF-Framework trägt in sich auch einige Einschränkungen. Die Methoden `getResourceProperty()` und `setResourceProperty()` werden ausschließlich für den Zugriff auf die Properties einer VO-Ressource, verwendet. Keine Unterelemente von den Properties sind einzeln aufrufbar.

Möchte man nun bei einer Person eine Information wie die Telefonnummer ändern, ist es erforderlich, die komplette Struktur aller Personen von der Ressource abzuholen. Man kann somit die entsprechende Information ändern und das komplette Informations-Paket wieder zurückschicken. In der VO wird dann die Property mit der neu gesetzten überschrieben.

Da dies nicht nur einen großen Traffic-Overhead bedeutet, sondern auch die Reinform einer Race-Condition ist (hat ein zweiter Verwalter zwischenzeitlich etwas geändert, werden seine Änderungen überschrieben), sollte hier eine adressierbare Änderung eingeführt werden.

Natürlich kann eine solche Funktion über proprietäre Methoden relativ einfach hinzugefügt werden, aber dies würde eine Abkehr vom Standard bedeuten. Vorzuziehen wäre es an dieser Stelle, den WSRF-Standard um eine entsprechende Funktion zu erweitern, die analog zur bereits existierenden Methode `queryResourceProperty()` funktioniert.

6.2.5 Validierung der Properties

Wie oben beschrieben, wird vom Client immer eine große Datenstruktur komplett neu gesetzt. Dabei garantiert die XML-Beschreibung des Resource Property Documents, dass die einzelnen Informationen vom gewünschten Datentypen sind. Allerdings werden hierbei keine logischen Zusammenhänge geprüft.

Es steht zu verhindern, dass beispielsweise zwei Personen doppelt hinzugefügt werden, oder dass die Referenz einer Person ungültig ist. Um Fehler bei der Validierung an den Client zu übermitteln, kann die WSRF-Spezifikation zur Fehlerbehandlung, die WS-BaseFaults [LiMe 06], verwendet werden.

6.2.6 Einführen eines Locking für Resouce Properties

Wie oben bereits geschrieben, entsteht durch das Lesen und Schreiben einer kompletten Resource Property eine Race Condition. Ein typischer Bearbeitungsschritt besteht aus dem Lesen einer Property, lokalem Ändern und dem „Zurück-Schreiben“ der Property. Wenn ein Dritter seinerseits parallel diese Schritte durchführt, hängt das Ergebnis der beiden Schreibvorgänge von der Reihenfolge ab. Der zuletzt Speichernde „gewinnt“, da keinerlei Sperrung durch einen Verwalter möglich ist.

Es sollte daher die Möglichkeit untersucht werden, eine Resource Property zur Bearbeitung zu sperren. Dies kann methodisch durch eine eigene Implementierung oder durch eine Erweiterung des WSRF, was sinnvoller wäre, erfolgen.

7 Zusammenfassung

Im Rahmen dieser Arbeit haben wir die Virtuelle Organisation als WS-Resource implementiert. Dabei werden nach einer Einführung in die Fragestellung, die grundlegenden wissenschaftlichen Erkenntnisse dargestellt.

Im Verlauf haben wir von der Fragestellung dann einen Anforderungskatalog, das im Laufe dieser Arbeit erfüllt wurde, abgeleitet. In diesem stellen wir Forderungen nicht nur an unsere Implementation, sondern allgemein an die Umsetzung Virtueller Organisation als WS-Resource. Dabei wurde auch die strittige Frage der Mitgliedschaft von Ressourcen geklärt. Durch die Aufnahme von WS-Resourcen, die im WSRF definiert sind, wurde ein – nach unserer Meinung – akzeptabler Kompromiss gefunden.

Daraufhin haben wir untersucht, welche aktuellen Projekte auch in ähnliche Richtungen forschen, oder eventuell eine ideale Ergänzung zu unserer geplanten Umsetzung darstellen würden. Dabei wurde ein Paper gefunden, in dem dieselbe Fragestellung gestellt wurde. Allerdings war dieses sehr kurz und lieferte nur Vorschläge, die teilweise mit unseren Ergebnissen übereinstimmten, aber keine Umsetzung. Daher haben wir einige Projekte betrachtet, die VOs aus einer anderen Motivation heraus implementiert haben. Einige dieser Projekte bieten sich als ergänzende Lösung zu der unseren an, beispielsweise um verteilte Authentifizierung zu erreichen (PERMIS).

Zur Umsetzung der geforderten Eigenschaften und Funktionen wurden dann anschließend die Anforderungen auf das WSRF abgebildet. Dabei haben wir eine Beschreibung der Eigenschaften einer VO als WS-Resource erstellt. Die Ergebnisse wurden wiederum in den Rahmen des GT4 eingepasst und mit Web Services zur Erzeugung und Bearbeitung versehen. Schließlich haben wir eine funktionsfähige Software-Implementierung der Anforderungen in Java geleistet. Um die Lauffähigkeit der erstellten Software zu demonstrieren, wurde ein einfacher Management-Client erzeugt. Dieser beweist, dass unsere Implementierung die im vorneherein gestellten Anforderungen erfüllt.

Bei der Implementierung hat sich die mangelnde Dokumentation des Globus Toolkits als großes Hindernis dargestellt. Es gibt zwar eine automatisch erzeugte API-Dokumentation, in dieser sind aber längst nicht alle Klassen und Methoden ausreichend beschrieben. Dadurch ging ein großer Teil der Bearbeitungszeit für die Suche nach Informationen bzw. auch für Trial-and-Error Programmierung verloren.

Schließlich haben wir jedoch eine funktionsfähige Lösung erzielt, bei der mittels eines einfachen Clients die als WS-Resourcen implementierten Virtuellen Organisationen gemanaged werden können. Die eben angesprochenen Einschränkungen sind dabei nicht negativ zu sehen, sondern bieten vielmehr Potential für weitere Entwicklungen.

Zusammenfassend bleibt zu sagen, dass wir einen weiteren Schritt in eine offensichtlich viel versprechende Richtung getan haben. Da wir aber längst nicht alle Aspekte des entsprechenden Spektrums abgedeckt haben, bleiben Virtuelle Organisationen weiterhin ein großes Feld für wissenschaftliche Forschungen.

Teil I

Appendix

A Abkürzungen

AC	Attribute Certificate
Ant	Another Neat Tool
API	Application Programming Interface
B2B	Business-to-Business
CA	Certificate Authority
CAS	Community Authorization Service
CORBA	Common Object Request Broker Architecture
DCE	Distributed Computing Environment
DN	Distinguished Name
DNS	Domain Name Server
DOM	Document Object Model
EDV	Elektronische Datenverarbeitung
EJB	Enterprise JavaBeans
EPR	End Point Reference
FCAPS	Faults, Configuration, Accounting, Performance, Security
FTP	File Transfer Protocol
GA	Gegenseitige Authetifizierung
GGF	Globus Grid Forum
GSI	Grid Security Infrastructure
GT4	Globus Toolkit 4
HLC	Hochleistungscomputer
HTTP	Hypertext Transfer Protocol
ID	Identifikationsnummer
IdP	Identity Provider
IdM	Identity Management
IP	Internet Protocol
ISSRG	Information Systems Security Research Group
JASS	Java Authentification and Authorisation Service
JNDI	Java Naming and Directory Interface
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
MACE	Middleware Architecture Committee for Education
MLS	Message Level Security
OASIS	Organization for the Advancement of Structured Information Standards
OGSA	Open Grid Services Architecture
OGSI	Open Grid Services Infrastructure
PC	Personal Computer
PDP	Policy Decision Point
PERMIS	Privilege and Role Management Infrastructure Standards Validation
PIP	Policy Information Point
PKI	Public Key Infrastructure
PMI	Privilege Management Infrastructure
RBAC	Role Based Access Control

RMI	Remote Method Invocation
RP	Resource Property
RPD	Resource Property Document
SAML	Security Assertion Markup Language
SAX	Simple API for XML
SP	Service Provider
QEPR	Qualified End Point Reference
QName	Qualified Name
SLA	Service Level Agreement
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
TCP	Transmission Control Protocol
TLS	Transport Level Security
UDDI	Universal Description, Discovery and Integration
URI	Uniform Resource Identifier
VO	Virtual Organization
VOMS	VO Membership Service
VOMRS	VO Membership Registration Service
WAN	Wide Area Network
WS	Web Service
WSDD	Web Services Deployment Descriptor
WSDL	Web Services Description Language
WSDL	Web Services Distributed Management
WSRF	Web Services Resource Framework
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XML-RPC	Extensible Markup Language Remote Procedure Control
XPath	XML Path Language

B WS-Standarde und -spezifikationen

BPEL4WS	Business Process Execution Language for Web Services
BPML	Business Process Modelling Language
CDL4WS	Choreography Definition Language for Web Services
MOWS	Management of Web Services
MUWS	Management Using Web Services
MTOM	SOAP Message Transmission Optimization Mechanism
SAML	Security Assertion Markup Language
SOAP	keine Bedeutung des Akronyms
SwA	SOAP with Attachments
UDDI	Universal Description, Discovery and Integration
WS-A	Web Services Addressing
WS-Attach	Web Services Attachments
WS-CAF	Web Services Composite Application Framework
WSCI	Web Services Choreography Interface
WSDL	Web Services Description Language
XML	eXtensible Markup Language
XOP	XML-binary Optimized Packaging

C Erforderliche Software bei der Installation des GT4

Die im folgenden genannten Software Bestandteile sind bei der Installation vom Globus Toolkit 4 erforderlich, falls es kein fertig installierbares Paket für das jeweilige System gibt:

- MAKE (GNU make). Mit dem Befehl `make --version` wird die entsprechende Version überprüft. Ähnlich wie Ant für Java Projekte kontrolliert und koordiniert `make` die Übersetzung vom Quellcode zu einem ausführbaren Programm.
- SUDO (sudo). Welche Version auf dem System läuft, ist mit Hilfe vom Befehl `sudo -V` rauszufinden. Mit Hilfe dieses Tools kann ein normaler Anwender Befehle im Berechtigungskontext eines anderen Nutzers, üblicherweise des Administrators, ausführen. Bei der Installation wird davon Gebrauch gemacht, um hier die Berechtigung des speziell angelegten Zugangs „*globus*“ einfließen zu lassen.
- Eine Datenbank [1] ist für optionale Globus-Kernelemente notwendig. Mit dem folgenden Commando `/etc/init.d/postgrsql start` wird die vorhandene Datenbank gestartet. Die Verwendung von Datenbanken wird in dieser Arbeit nicht eingegangen, da für die Implementation keine benötigt wird.
- C Compiler (gcc, Version 3.2 soll vermieden werden). Mit dem Befehl `gcc --version` kann die gcc-Version angezeigt werden.
- TAR (GNU tar). Um die tar-Version rauszufinden, wird der Befehl `tar --version` angegeben. Tar ist notwendig, um komprimierte Archive von Quellen auszupacken. Allerdings könnte es auch notwendig sein, um eine Binärversion vom Globus auszupacken und zu installieren – dies hängt von dem jeweiligen Zielsystem ab.
- SED (GNU sed). Die installierte Version des Stream Editors wird mit Hilfe vom Befehl `sed --version` angezeigt. Dieser wird beim Kompilieren verwendet, um Informationen aufzubereiten.
- ZLIB (zlib 1.1.4+) ist eine Kompressionsbibliothek, die zur Grundausstattung der meisten Systeme gehören dürfte.

[1] JDBC fähige Datenbank, wie PostgreSQL 7.1+ notwendig

D Web Services Deployment Descriptor (WSDD)

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment name="defaultServerConfig"
    xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <!-- Instance service -->
    <service name="cojocaru/VO/VOInstanceService" provider="Handler" use="literal" style="document"
        >
        <parameter name="className" value="cojocaru.VO.impl.VOInstanceService"/>
        <wsdlFile>share/schema/cojocaru/VO/VOInstanceService_instance/VOInstanceService_service.
            wsdl</wsdlFile>
        <parameter name="allowedMethods" value="*"/>
        <parameter name="handlerClass" value="org.globus.axis.providers.RPCProvider"/>
        <parameter name="scope" value="Application"/>
        <parameter name="providers" value="GetRPProvider GetMRPProvider SetRPProvider
            QueryRPProvider"/>
        <parameter name="loadOnStartup" value="true"/>
    </service>

    <!-- Factory service -->
    <service name="cojocaru/VO/VOFactoryService" provider="Handler" use="literal" style="document">
        <parameter name="className" value="cojocaru.VO.impl.VOFactoryService"/>
        <wsdlFile>share/schema/cojocaru/VO/VOFactoryService_instance/VOFactoryService_service.wsdl<
            /wsdlFile>
        <parameter name="allowedMethods" value="*"/>
        <parameter name="handlerClass" value="org.globus.axis.providers.RPCProvider"/>
        <parameter name="scope" value="Application"/>
        <parameter name="instance" value="cojocaru/VO/VOInstanceService"/>
    <parameter name="loadOnStartup" value="true"/>
    </service>

</deployment>
```

E Kompilieren von Web Services

```
lxvm1:~/Globus_Impl # ./globus-build-service.sh VO
Buildfile: build.xml

init:
   [delete] Deleting directory /root/Globus_Impl/build/lib
       [mkdir] Created dir: /root/Globus_Impl/build/lib
flatten:

WSDLUptodate:

flatten:

generateBindings:

bindingUptodate:

generateBinding:

stubs:

mergePackageMapping:
      [echo] Merging /root/Globus_Impl/namespace2package.mappings

generateStubs:
      [echo]  Generating stubs from VOInstanceService_service.wsdl
      [java] {http://schemas.xmlsoap.org/ws/2004/03/addressing}Action
           already exists

factoryFlatten:

WSDLUptodate:

flatten:

generateFactoryBindings:

bindingUptodate:

generateBinding:

factoryStubs:

mergePackageMapping:
     [echo] Merging /root/Globus_Impl/namespace2package.mappings

generateStubs:
     [echo]  Generating stubs from VOFactoryService_service.wsdl
     [java] {http://schemas.xmlsoap.org/ws/2004/03/addressing}Action
            already exists

compileStubs:
     [javac] Compiling 20 source files to
```

```
        /root/Globus_Impl/build/stubs-cojocaru_VO/classes
     [javac] Note: Some input files use unchecked or unsafe
         operations.
     [javac] Note: Recompile with -Xlint:unchecked for
         details.
        [copy] Copying 20 files to
        /root/Globus_Impl/build/stubs/classes

jarStubs:
      [jar] Building jar:
       /root/Globus_Impl/build/lib/cojocaru_VO_stubs.jar

compile:

jar:
      [jar] Building jar: /root/Globus_Impl/build/lib/cojocaru_VO.jar

copyJars:

dist:

makeGar:

testJars:

copyJars:
       [copy] Copying 2 files to /root/Globus_Impl/tmp/gar/lib

testSchema:

copySchema:
       [copy] Copying 8 files to /root/Globus_Impl/tmp/gar/schema

testEtc:

copyEtc:
     [copy] Copied 1 empty directory to 1 empty directory under
     /root/Globus_Impl/tmp/gar/etc
       [antcall] Parent project doesn't contain any reference '${garshare.id}'
testShare:

copyShare:
         [antcall] Parent project doesn't contain any reference '${gardocs.id}'

testDocs:

copyDocs:
         [antcall] Parent project doesn't contain any reference '${garbin.id}'

testBin:

copyBin:
       [copy] Copying 1 file to /root/Globus_Impl/tmp/gar
       [copy] Warning: Could not find file
            /root/Globus_Impl/cojocaru/VO/deploy-client.wsdd to copy.
       [copy] Copying 1 file to /root/Globus_Impl/tmp/gar
       [jar] Building jar: /root/Globus_Impl/cojocaru_VO.gar
       [delete] Deleting directory /root/Globus_Impl/tmp/gar
BUILD SUCCESSFUL
Total time: 7 seconds
```

F Deployment

```
globus@lxvm1:~> /usr/local/globus-4.0.4/bin/globus-deploy-gar
/root/Globus_Impl/cojocaru_VO.gar
Deploying gar file...

  Deploying gar with profile: <default>
    Created dir: /usr/local/globus-4.0.4/share/globus_wsrf_common/tmp/gar
    Skipping fileset for directory
    /usr/local/globus-4.0.4/share/globus_wsrf_common/tmp. It is empty.
    Created dir:
    /usr/local/globus-4.0.4/share/globus_wsrf_common/tmp/gar/schema
     Created dir:
     /usr/local/globus-4.0.4/share/globus_wsrf_common/tmp/gar/etc
     Created dir:
    /usr/local/globus-4.0.4/share/globus_wsrf_common/tmp/gar/bin
     Created dir:
     /usr/local/globus-4.0.4/share/globus_wsrf_common/tmp/gar/docs
     Created dir:
    /usr/local/globus-4.0.4/share/globus_wsrf_common/tmp/gar/share
     Created dir:
    /usr/local/globus-4.0.4/share/globus_wsrf_common/tmp/gar/lib
     Expanding: /root/Globus_Impl/cojocaru_VO.gar into
    /usr/local/globus-4.0.4/share/globus_wsrf_common/tmp/gar
     Copying 8 files to /usr/local/globus-4.0.4/share/schema
     Skipping fileset for directory
    /usr/local/globus-4.0.4/etc/cojocaru_VO. It is empty.
     Skipping fileset for directory
    /usr/local/globus-4.0.4. It is empty.
     Skipping fileset for directory
    /usr/local/globus-4.0.4/bin. It is empty.
     Copying 2 files to /usr/local/globus-4.0.4/lib
     Skipping fileset for directory
    /usr/local/globus-4.0.4/share/licenses. It is
     empty.
     deploying server config...
     Copying 1 file to
    /usr/local/globus-4.0.4/etc/cojocaru_VO
     deploying JNDI config...
     Copying 1 file to
    /usr/local/globus-4.0.4/etc/cojocaru_VO
     Deleting directory
    /usr/local/globus-4.0.4/share/globus_wsrf_common/tmp/gar

Deploy successful
```

G Starten vom Globus-Container

```
bus@lxvm1:~> globus-start-container -nosec
Starting SOAP server at: http://192.168.218.21:8080/wsrf/services/
With the following services:

[1]: http://192.168.218.21:8080/wsrf/services/AdminService
[2]: http://192.168.218.21:8080/wsrf/services/AuthzCalloutTestService
[3]: http://192.168.218.21:8080/wsrf/services/CASService
[4]: http://192.168.218.21:8080/wsrf/services/ContainerRegistryEntryService
[5]: http://192.168.218.21:8080/wsrf/services/ContainerRegistryService
[6]: http://192.168.218.21:8080/wsrf/services/CounterService
[7]: http://192.168.218.21:8080/wsrf/services/DefaultIndexService
[8]: http://192.168.218.21:8080/wsrf/services/DefaultIndexServiceEntry
[9]: http://192.168.218.21:8080/wsrf/services/DefaultTriggerService
[10]: http://192.168.218.21:8080/wsrf/services/DefaultTriggerServiceEntry
[11]: http://192.168.218.21:8080/wsrf/services/DelegationFactoryService
[12]: http://192.168.218.21:8080/wsrf/services/DelegationService
[13]: http://192.168.218.21:8080/wsrf/services/DelegationTestService
[14]: http://192.168.218.21:8080/wsrf/services/InMemoryServiceGroup
[15]: http://192.168.218.21:8080/wsrf/services/InMemoryServiceGroupEntry
[16]: http://192.168.218.21:8080/wsrf/services/InMemoryServiceGroupFactory
[17]: http://192.168.218.21:8080/wsrf/services/IndexFactoryService
[18]: http://192.168.218.21:8080/wsrf/services/IndexService
[19]: http://192.168.218.21:8080/wsrf/services/IndexServiceEntry
[20]: http://192.168.218.21:8080/wsrf/services/ManagedExecutableJobService
[21]: http://192.168.218.21:8080/wsrf/services/ManagedJobFactoryService
[22]: http://192.168.218.21:8080/wsrf/services/ManagedMultiJobService
[23]: http://192.168.218.21:8080/wsrf/services/ManagementService
[24]:
http://192.168.218.21:8080/wsrf/services/NotificationConsumerFactoryService
[25]: http://192.168.218.21:8080/wsrf/services/NotificationConsumerService
[26]: http://192.168.218.21:8080/wsrf/services/NotificationTestService
[27]:
http://192.168.218.21:8080/wsrf/services/PersistenceTestSubscriptionManager
[28]: http://192.168.218.21:8080/wsrf/services/RendezvousFactoryService
[29]: http://192.168.218.21:8080/wsrf/services/SampleAuthzService
[30]: http://192.168.218.21:8080/wsrf/services/SecureCounterService
[31]: http://192.168.218.21:8080/wsrf/services/SecurityTestService
[32]: http://192.168.218.21:8080/wsrf/services/ShutdownService
[33]: http://192.168.218.21:8080/wsrf/services/SubscriptionManagerService
[34]: http://192.168.218.21:8080/wsrf/services/TestAuthzService
[35]: http://192.168.218.21:8080/wsrf/services/TestRPCService
[36]: http://192.168.218.21:8080/wsrf/services/TestService
[37]: http://192.168.218.21:8080/wsrf/services/TestServiceRequest
[38]: http://192.168.218.21:8080/wsrf/services/TestServiceWrongWSDL
[39]: http://192.168.218.21:8080/wsrf/services/TriggerFactoryService
[40]: http://192.168.218.21:8080/wsrf/services/TriggerService
[41]: http://192.168.218.21:8080/wsrf/services/TriggerServiceEntry
[42]: http://192.168.218.21:8080/wsrf/services/Version
[43]: http://192.168.218.21:8080/wsrf/services/WidgetNotificationService
[44]: http://192.168.218.21:8080/wsrf/services/WidgetService
[45]: http://192.168.218.21:8080/wsrf/services/cojocaru/VO/VOFactoryService
[46]: http://192.168.218.21:8080/wsrf/services/cojocaru/VO/VOInstanceService
```

```
[47]: http://192.168.218.21:8080/wsrf/services/gsi/AuthenticationService
[48]: http://192.168.218.21:8080/wsrf/services/mds/test/execsource/IndexService
[49]:
http://192.168.218.21:8080/wsrf/services/mds/test/execsource/IndexServiceEntry
[50]: http://192.168.218.21:8080/wsrf/services/mds/test/subsource/IndexService
[51]:
http://192.168.218.21:8080/wsrf/services/mds/test/subsource/IndexServiceEntry
2007-06-11 09:39:35,722 INFO  impl.DefaultIndexService
[ServiceThread-10,processConfigFile:107] Reading default registration
configuration from file:
/usr/local/globus-4.0.4/etc/globus_wsrf_mds_index/hierarchy.xml
```

H JNDI Deployment File

```
<?xml version="1.0" encoding="UTF-8"?>
<jndiConfig xmlns="http://wsrf.globus.org/jndi/config">

<!-- Instance service -->
<service name="cojocaru/VO/VOInstanceService">
    <resource name="home" type="cojocaru.VO.impl.VOResourceHome">
    <resourceParams>

        <parameter>
            <name>resourceClass</name>
            <value>cojocaru.VO.impl.VOResource</value>
        </parameter>

        <parameter>
            <name>resourceKeyType</name>
            <value>java.lang.Integer</value>
        </parameter>

        <parameter>
            <name>resourceKeyName</name>
            <value>{http://www.cojocaru.de/Namespaces/VO/
                    VOInstanceService_instance}VOResourceKey</value>
        </parameter>

        <parameter>
            <name>factory</name>
            <value>org.globus.wsrf.jndi.BeanFactory</value>
        </parameter>

        <parameter>
            <name>instanceServicePath</name>
            <value>cojocaru/VO/VOInstanceService</value>
        </parameter>

    </resourceParams>
    </resource>
</service>

<!-- Factory service -->
<service name="cojocaru/VO/VOFactoryService">
    <!-- Um mehrmalige Angabe dieser Parameter zu vermeiden,
    weil die IS und FS dasselbe RH haben,verwenden wir einfach ein <resourceLink>
        <resourceLink name="instanceHome" target="java:comp/env/services/
                            cojocaru/VO/VOInstanceService/home"/>
</service>

</jndiConfig>
```

I Beispiel für eine Endpoint Reference

```
<ns1:VOResourceReference
        xsi:type="ns2:EndpointReferenceType"
        xmlns:ns1="http://www.cojocaru.de/Namespaces/VO/VOInstanceService"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:ns2="http://schemas.xmlsoap.org/ws/2004/03/addressing">
    <ns2:Address xsi:type="ns2:AttributedURI">
        http://127.0.0.1:8080/wsrf/services/cojocaru/VO/VOInstanceService
    </ns2:Address>
    <ns2:ReferenceProperties xsi:type="ns2:ReferencePropertiesType">
        <ns1:VOResourceKey
    xmlns:ns1="http://www.cojocaru.de/Namespaces/VO/VOInstanceService_instance">
            23775954
        </ns1:VOResourceKey>
    </ns2:ReferenceProperties>
    <ns2:ReferenceParameters xsi:type="ns2:ReferenceParametersType"/>
</ns1:VOResourceReference>
```

Literaturverzeichnis

[Abbe 97] ABBE, MOWSHOWITZ: *Virtual Organization. In: Communications of the ACM 40 (1997).* September 1997. Nr. 9, S.30-37.

[ABC+ 98] APPARAO, VIDUR, STEVE BYRNE, MIKE CHAMPION, SCOTT ISAACS, IAN JACOBS, ARNAUD LE HORS, GAVIN NICOL, JONATHAN ROBIE, ROBERT SUTOR, CHRIS WILSON und LAUREN WOOD: *Document Object Model (DOM) Level 1 Specification, Version 1.0.* W3C, Oktober 1998, http://www.w3.org/TR/REC-DOM-Level-1 .

[ACC+ 04] ALFIERI, R., R. CECCHINI, V. CIASCHINI, F. SPATARO, K. L'ÖRENTEY, A. FROHNER und L. DELL'AGNELO: *From gridmap-file to VOMS: managing Authorization in a Grid environment.* April 2004, http://grid-auth.infn.it/docs/voms-FGCS.pdf .

[AkAl 07] AKRAM, ASIF und ROB ALLAN: *WSRF based Virtual Organization Middleware.* In: *18th Annual IRMA International Conference, Vancouver, British Columbia, Canada*, May 2007, http://pubs.doc.ic.ac.uk/WSRF-Virtual-Organization/ .

[Akra 06] AKRAM, ASIF: *Manage multiple resources with a single instance service.* August 2006, http://www.ibm.com/developerworks/edu/gr-dw-gr-mltressis.html?ca=drs- .

[BDH+ 06] BAROZZI, NICOLA KEN, PETER DONALD, ERIK HATCHER, SAM RUBY, STEPHANE BAILLIEZ, JON S. STEVENS und MARTIJN KRUITHOF: *Apache Ant User Manual.* 2006, http://ant.apache.org/manual/index.html .

[BHF 03] BERMAN, F., A. HEY und G. FOX: *Grid Computing: Making The Global Infrastructure a Reality.* Willey, 2003.

[BHM+ 04] BOOTH, DAVID, HUGO HAAS, FRANCIS MCCABE, ERIC NEWCOMER, MICHAEL CHAMPION, CHRIS FERRIS und DAVID ORCHARD: *Web Services Architecture.* W3C, Februar 2004, http://www.w3.org/TR/ws-arch/ .

[BMW 06] BULLARD, VAUGHN, BRYAN MURRAY und KIRK WILSON: *An Introduction to WSDM.* OASIS, Februar 2006, http://docs.oasis-open.org/wsdm/wsdm-1.0-intro-primer-cd-01.pdf .

[Brow 02] BROWNELL, DAVID: *Sax2. Processing XML efficiently with Java.* O'Reilly, ISBN 13: 9780596002374, Januar 2002.

[Chad 01] CHADWICK, DAVID: *An X.509 Role Based Privilege Management Infrastructure.* In: *Briefing - Global InfoSecurity 2002, World Markets Research Centre Ltd.* October 2001, http://www.cs.kent.ac.uk/pubs/2001/2125 . On accompanying CD-ROM Reference Library/03.pdf.

[ChOt 02] CHADWICK, DAVID und ALEXANDER OTENKO: *The PERMIS X.509 Role Based Privilege Management Infrastructure.* Science BV, Dezember 2002. Future Generation Computer Systems.

[ChOt 03] CHADWICK, DAVID und ALEXANDER OTENKO: *The PERMIS X.509 role based privilege management infrastructure.* Future Gener. Comput. Syst., 19(2):277–289, 2003.

[CHvRR 04] CLEMENT, LUC, ANDREW HATELY, CLAUS VON RIEGEN und TONY ROGERS: *UDDI Version 3.0.2.* OASIS, Oktober 2004, http://uddi.org/pubs/uddi_v3.htm .

[CKM+ 03] CATANIA, NICOLAS, PANKAJ KUMAR, BRYAN MURRAY, HOMAYOUN POURHEDAR, KLAUS WURSTER und WILLIAM VAMBENEPE: *Web Services Management Framework - Overview (WSMF-Overview) Version 2.0.* Hewlett-Packard Development Company L.P, 2003, http://xml.coverpages.org/WSMF-Overview.pdf .

[ClDe 99] CLARK, JAMES und STEVE DEROSE: *XML Path Language XPath*. W3C, November 1999, http://www.w3.org/TR/1999/REC-xpath-19991116 .

[Cohe 01] COHEN, FRANK: *Myths and misunderstandings surrounding SOAP*. September 2001, http://www.ibm.com/developerworks/webservices/library/ws-spmyths.html .

[DaMa 93] DAVIDOW, WILLIAM H. und MICHAEL S. MALONE: *The Virtual Cooperation*. HarperBusiness, U.S., September 1993.

[Dan 06] DAN JEMIOLO (IBM): *Web Services Resource Metadata 1.0 (WS-ResourceMetadataDescriptor)*. OASIS, November 2006, http://docs.oasis-open.org/wsrf/wsrf-ws_resource_metadata_descriptor-1.0-spec-cs-01.pdf .

[FKS+ 06] FOSTE, I., H. KISHIMOTO, A. SAVVA, D. BERRY, A. DJAOUI, A. GRIMSHAW, B. HORN, F. MACIEL, F. SIEBENLIST, R. SUBRAMANIAM, J. TREADWELL und J. VON REICH: *The Open Grid Services Architecture, Version 1.5*. Open Grid Forum, July 2006, http://www.ogf.org/documents/GFD.80.pdf .

[FoKe 01] FOSTER, I. und C. KESSELMAN: *The Anatomy of the Grid: Enabling Scalable Virtual Organisation*. 2001, http://www.globus.org/alliance/publications/papers/anatomy.pdf .

[FoKe 04] FOSTER, I. und C. KESSELMAN: *The Grid: Blueprint for a new computing infrastructure*. Morgan Kaufmann, August 2004.

[Fost 02] FOSTER, IAN: *What is the Grid? A Three Point Checklist*. Juli 2002, http://www-fp.mcs.anl.gov/ foster/Articles/WhatIsTheGrid.pdf . Argonne National Laboratory & University of Chicago.

[FrAn 05] FREEMAN, TIM und RACHANA ANANTHAKRISHNAN: *Authorization processing for Globus Toolkit Java Web Services*. 2005, http://www-128.ibm.com/developerworks/grid/library .

[GHM 06] GRAHAM, STEVE, DAVID HULL und BRYAN MURRAY: *Web Services Base Notification 1.3*. OASIS, October 2006, http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf .

[GHR 06] GUDGIN, MARTIN, MARC HADLEY und TONY ROGERS: *Web Services Addressing 1.0 - Core*. OASIS, Mai 2006, http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/ .

[GKM+ 06] GRAHAM, STEVE, ANISH KARMARKAR, JEFF MISCHKINSKY, IAN ROBINSON und IGOR SEDUKHIN: *Web Services Resource 1.2*. OASIS, April 2006, http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf .

[Glob 05] GLOBUS PROJECT: *Globus Toolkit 4.0.4 Javadocs*. 2005, http://www-unix.globus.org/api/javadoc-4.0/ .

[GrTr 06] GRAHAM, STEVE und JEM TREADWELL: *Web Services Resource Properties 1.2*. OASIS, April 2006, http://docs.oasis-open.org/wsrf/wsrf-ws_resource_properties-1.2-spec-os.pdf .

[Hais 03] HAISCHT, DANIEL S.: *Kits und Packs. Die Web Services Toolkits von Sun und IBM im Vergleich*. XML Magazin, April 2003, http://xml-magazin.de/itr/online_artikel/psecom,id,241,nodeid,69.html .

[HAN 99] HEGERING, HEINZ-GERD, SEBASTIAN ABECK und BERNHARD NEUMAIR: *Integriertes Management vernetzter Systeme. Konzepte, Architekturen und deren betrieblicher Einsatz*. Februar 1999.

[He 03] HE, HAO: *What Is Service-Oriented Architecture*. O'reilly xml.com, September 2003, http://www.xml.com/pub/a/ws/2003/09/30/soa.html .

[IBM 07] IBM CORPORATION: *Funktionsweise: World Community Grid*. Februar 2007, http://www-05.ibm.com/de/pov/grid/HIW_02132007_tr_de.pdf .

[JBC+ 06] JENDROCK, ERIC, JENNIFER BALL, DEBBIE CARSON, IAN EVANS, SCOTT FORDIN und KIM HAASE: *The Java EE 5 Tutorial*. Addison-Wesley, 2006, http://java.sun.com/javaee/5/docs/tutorial/doc/JavaEETutorial.pdf . Third Edition.

[Jeck 04] JECKLE, MARIO: *Was ist die eXtensible Markup Language? (Tutorial).* 2004, http://www.jeckle.de/xml/index.html .

[Kemp 04] KEMP, JOHN: *Liberty ID-WSF a Web Services Framework.* 2004, http://www.projectliberty.org/liberty/content/download/390/2729/file/Liberty_ID-WSF_Web_Services_Framework.pdf .

[KMC+ 05] KEMP, JOHN, PRATEEK MISHRA, SCOTT CANTOR, EVE MALER und ROB PHILPOTT: *Authentication Context for the OASIS Security Assertion Markup Language (SAML) V2.0.* OASIS, März 2005, http://docs.oasis-open.org/security/saml/v2.0/ .

[Kner 06] KNERR, TORBEN: *Direkter Zugriff auf Grid Services (Workshop Advanced Middleware).* Juni 2006, http://mobdev.tknerr.de/wp-content/uploads/2006/09/direct-grid-web-service-access-with-mobile-devices_v7_final_draft.pdf . Seite 13-15.

[Korn 04] KORNMAYER, HARALD: *Vernetzte Welten. Das Globus-Toolkit, Version 2.* Linux Magazin, Ausgabe 06/04, Juni 2004, http://www.linux-magazin.de/Artikel/ausgabe/2004/06/globus/globus.html .

[Lee 02] LEE, ROSANNA: *The JNDI Tutorial.* November 2002, http://java.sun.com/products/jndi/tutorial/index.html .

[LiMe 06] LIU, LILY und SAM MEDER: *Web Services Base Faults 1.2.* OASIS, April 2006, http://docs.oasis-open.org/wsrf/wsrf-ws_base_faults-1.2-spec-os.pdf .

[Manh 06] MANHART, DR. KLAUS: *Grid-Technologie Teil 2: Grundlagen, Dienste und Tools.* May 2006, http://www.tecchannel.de/index.cfm?&pk=438803 .

[McKe 07] MCKENDRICK, JOE: *From Clustering to Grid: A New Data Infrastructure Emerges.* April 2007, http://www.ioug.org/tech/articles/ResearchWire-ClusteringtoGrid.pdf .

[MiLa 07] MITRA, NILO und YVES LAFON: *SOAP Version 1.2 Part 0: Primer (Second Edition).* W3C, April 2007, http://www.w3.org/TR/2007/REC-soap12-part0-20070427/ .

[MSB 06] MAGUIRE, TOM, DAVID SNELLING und TIM BANKS: *Web Services Service Group 1.2.* OASIS, April 2006, http://docs.oasis-open.org/wsrf/wsrf-ws_service_group-1.2-spec-os.pdf .

[Müll 97] MÜLLER, THOMAS: *Virtuelle Organisation Konzept, Theoriebasis, Möglichkeiten und Grenzen.* 1997, http://w3.ub.uni-constanz.de/v13/volltexte/1999/293//pdf/293_1.pdf . Schriftenreihe Management Forschung und Praxis der Universität Konstanz, Band 21.

[Open 98] OPENLDAP PROJECT: *A Quick-Start Guide.* August 1998, http://www.openldap.org/doc/admin23/quickstart.html .

[Open 00] OPENCA TEAM: *The Opensource PKI Book. A guide to PKIs and Opensource implementations (Version 2.4.6).* Symeon (Simos) Xenitellis, 2000, http://ospkibook.sourceforge.net/ .

[Prof 05] PROF. DR. CH. REICH: *Grid Computing (Gitterberechung).* Mai 2005, http://www.unfug.org/data/foils/2005/GridComputing.pdf . UnFUG Vortrag.

[Proj 05] PROJEKT, D-GRID: *Thesenpapier zum VO Management in D-Grid.* Oktober 2005, http://www.d-grid.de/fileadmin/dgrid_document/Dokumente/VOMS-Thesenpapier.pdf . Management von Virtuellen Organisationen (Workshop).

[PWK+ 02] PEARLMAN, LAURA, VON WELCH, CARL KESSELMAN, IAN FOSTER und STEVEN TUECKE: *A Community Authorization Service for Group Collaboration.* 2002, http://www.globus.org/alliance/publications/papers/CAS_2002_Revised.pdf .

[ReSc 04] REINEFELD, ALEXANDER und FLORIAN SCHINTKE: *Dienste und Standards fur das Grid Computing.* 2004, http://www.zib.de/CSR/Publications/2004-reinefeld-lni.pdf .

[Rose 98] ROSENBERRY, WARD: *DCE Today. An indispensable guide.* Catalog number G604P, ISBN 0135756979, Juli 1998. A source book from The Open Group.

[RPW 03] REICHWALD, RALF, ARNOLD PICOT und ROLF T. WIGAND: *Die grenzenlose Unternehmung. Information, Organisation und Management*. Gabler Verlag, 5. Auflage, ISBN-13: 978-3409522144, Januar 2003.

[RöSc 05] RÖHRL, ARMIN und STEFAN SCHMIEDL: *Letś Grid! Ein Überblick über Grid Computing und das Globus Toolkit.* Linux Magazin, Ausgabe 08/04, 2005, http://linuxenterprise.de/itr/online_artikel/psecom,id,576,nodeid,9.html .

[SBB+ 06] SCAVO, TOM, TOM BARTON, JIM BASNEY, TIM FREEMAN, FRANK SIEBENLIST, VON WELCH, RACHANA ANANTHAKRISHNAN, BILL BAKER, MONTE GOODE und KATE KEAHEY: *Identity Federation and Attribute-based Authorization through the Globus Toolkit, Shibboleth, GridShib, and MyProxy*. April 2006, http://grid.ncsa.uiuc.edu/papers/gridshib-pki06-final.pdf . The 5th Annual PKI Workshop.

[Schi 07] SCHIFFERS, MICHAEL: *Management dynamischer Virtueller Organisationen in Grids*. Dissertation, Ludwig–Maximilians–Universität München, 2007.

[ScSt 07] SCHOLZ, UNIV.-PROF. DR. CHRISTIAN und DR. VOLKER STEIN: *Kurzfassung: Virtualisierung*. 2007, http://www.org-portal.org/portal/artikel.php?did=317 .

[Sess 03] SESSELMANN, THOMAS: *Das Grid. Einordnung einer Technologie für verteiltes Rechnen (Hauptseminar)*. 2003, http://eris.prakinf.tu-ilmenau.de/edu/HS/SS2003/Sesselmann03Grid.pdf .

[SFA 07] SIEBENLIST, FRANK, TIM FREEMAN und RACHANA ANANTHAKRISHNAN: *Globus Toolkit Java Authorization Framework*. 2007, http://www.globus.org/toolkit/docs/development/4.2-drafts/security/authzframe/gtJavaAuthzEngine.pdf .

[Shib 01] SHIBBOLETH WORKING GROUP: *Shibboleth - Specification - DRAFT v1.0*. Mai 2001, http://shibboleth.internet2.edu/docs/draft-internet2-shibboleth-specification-00.html#abstract .

[SKSW 06] SINNOT, R.O., J. KOETSIER, A.J. STELL und J. WATT: *DyVOSE Project: Experiences in Applying Privilege Management Infrastructures*. 2006, http://labserv.nesc.gla.ac.uk/projects/dyvose/publicity/AHM2006JPW_FINAL.doc . National e-Science Centre, University of Glasgow.

[SoCh 06] SOTOMAYOR, BORJA und LISA CHILDERS: *Globus Toolkit 4. Programming Java-Services*. Morgan Kaufmann Publishers, ISBN 13:978-0-12-369404-1, 2006.

[Soto 05] SOTOMAYOR, BORJA: *The Globus Toolkit 4 Programmer's Tutorial*. November 2005, http://gdp.globus.org/gt4-tutorial/download/progtutorial-pdf_0.2.1.tar.gz . University of Chicago Department of Computer Science.

[SrBa 06] SRINIVASAN, LATHA und TIM BANKS: *Web Services Resource Lifetime 1.2*. OASIS, April 2006, http://docs.oasis-open.org/wsrf/wsrf-ws_resource_lifetime-1.2-spec-os.pdf .

[Sun 03] SUN DEVELOPER NETWORK: *Remote Method Invocation - Distributed Computing for Java*. 2003, http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp . SDN.

[Sun 06] SUN DEVELOPER NETWORK: *Enterprise JavaBeans Technology*. 2006, http://java.sun.com/products/ejb/index.jsp . SDN.

[SZM 06a] SCHIFFERS, MICHAEL, WOLFGANG ZIEGLER und JENS-MICHAEL MILKE: *Rahmenkonzept fitr das Management virtueller Organisationen in D-Grid*. 2006, https://www.d-grid.de/fileadmin/user_upload/documents/DGI-FG1-10/VO_Rahmenkonzept_0.9.pdf .

[SZM 06b] SCHIFFERS, MICHAEL, WOLFGANG ZIEGLER und JENS-MICHAEL MILKE: *Virtuelle Organisation in Grids: Charakterisierung und Management*. 2006.

[TCF+ 03] TUECKE, S., K. CZAJKOWSKI, I. FOSTER, J. FREY, S. GRAHAM, C. KESSELMAN, T. MAQUIRE, T. SANDHOLM, T. SANDHOLM, D. SNELLING und P. VANDERBILT: *Open Grid Services Infrastructure (OGSI) Version 1.0*. OASIS, Juni 2003, http://www.ggf.org/ogsi-wg .

[The 06] THE APACHE SOFTWARE FOUNDATION: *WebServices - Axis (Version 1.2)*. 2006, http://ws.apache.org/axis/java/user-guide.pdf .

[Thor 05] THORSTEN HORN (W3C): *Web Services mit SOAP, WSDL und UDDI.* 2005, http://www.torsten-horn.de/techdocs/soap.htm#WebServices .

[Tim 05] TIM BANKS (IBM): *Web Services Resource Framework (WSRF) - Primer.* OASIS, Dezember 2005, http://docs.oasis-open.org/wsrf/wsrf-primer-1.2-primer-cd-01.pdf .

[uDTB 07] DR. TONI BOLLINGER, PROF. DR. MICHAEL GERNDT UND: *Grid Computing fltr Business-Intelligence-Anwendungen.* EPR Manager, July 2007, http://www.erpmanager.de/magazin/artikel_1520-610_grid_computing_business_intelligence.html .

[VOX 04] VOX PROJECT TEAM: *VOMRS User Guide.* Fermi National Accelerator Laboratory, Computing Division, 2004, http://computing.fnal.gov/docs/products/vomrs/pdf/vox.pdf . Chicago, USA.

[Wall 07] WALLNAU, KURT: *Common Object Request Broker Architecture.* Januar 2007, http://www.sei.cmu.edu/str/descriptions/corba.html . Software Engeneering Institut, Carnegie Mellon University.

[WK 06] W3C-KONSORTIUM: *Web Services Description Language (WSDL) Version 2.0 Part 0: Primer.* 2006, http://www.w3c.org/TR/wsdl20-primer .

[Zimm 80] ZIMMERMANN, HUBERT: *OS1 Reference Model - The IS0 Model of Architecture for Open Systems Interconnection (OSI).* IEEE TRANSACTIONS ON COMMUNICATIONS, VOL.COM-28, NO.4, April 1980.

Index

Access-Manager, 37
Another Neat Tool (Ant), 67
Anpassungsphase, 26
Apache SOAP, 65
Application Layer, 22
Auflösungsphase, 26
Authentifizierung, 25
Autorisierung, 25

BaseFaultType, 14
Benutzerkonten, 36
Betriebsphase, 26
build.mapping, 67

className, 70
Collective Layer, 22
Common parameters, 70
Community Authorization Service (CAS), 38
Connectivity Layer, 21
createVO(), 51, 57

Data Management, 19
DeleteResourceProperty, 13, 18
deploy-server.wsdd, 67
Deployment, 68
destroy-Methode, 13
Distributed computing, 8
DOM-Parser, 67

Endpoint Reference (EPR), 12

Fabric Layer, 21
Formationsphase, 26

get*, 55
GetCurrentTime, 58
getId, 60
GetInstanceVOResourceHome, 56, 57
GetMultipleResourceProperties, 17, 58
GetResourceProperty, 16, 58
GetResourcePropertyDocument, 18
GetTerminationTime, 58
GetVOList(), 56, 57
Globus Service Build Tools (GSBT), 67
Globus Toolkit 4 (GT4), 5, 19, 65
globus-build-service.sh, 67, 72
Globus-Container, 68
Grid Computing, 1, 20
Grid Security Infrastructure (GSI), 19
Grid-Architektur, 21
Grid-Infrastruktur, 21
Grid-Ressource, 36
GT4 Execution Management, 19
GT4 Security, 19

Identity Management (IdM), 38
Identity Provider (IdP), 38
Information Service, 19
Information Systems Security Research Group (ISSRG), 36
InsertResourceProperties, 18
isPermitted(), 62

Java Naming and Directory Interface (JNDI), 70
jndiConfig, 71

Lebenszyklus, 2, 25
Lebenszyklusmanagement, 25
Liberty, 37
Liberty Alliance, 37
loadOnStartup, 70
Lokalisierungsservice, 38

managed objects, 25
Management von Mitgliedschaften, 25
Management-Anforderung, 26
Mapping, 71
MembershipContentRules, 14
Middleware Architecture Committee for Education (MACE), 38
Multiple Resource Web Service, 9
Multiple Ressource, 42

Notification consumer, 15
Notification producer, 15

Open Grid Services Architecture (OGSA), 1
Open Grid Services Infrastructure (OGSI), 12

PERMIS, 35
Planungsphase, 26
Policy Decision Point (PDP), 42, 54
Policy Information Point (PIP), 42, 54
Principal, 42
Provider, 57
Proxy-Zertifikat, 36
PutResourcePropertyDocument, 13, 18

Qualified Name (QName), 17, 43
QueryResourceProperties, 13, 18, 58

ReflectionResourceProperty, 60
RemoteException, 43
Resource Layer, 21
Resource Property Document (RPD), 13, 15, 16, 49
Ressource Property, 15
Ressourcen-Identifikator, 60
Ressourcen-Initialisierung, 61

SAX-Parser, 67
securityDescriptor, 70
Service data values, 15
Service Description, 10
Service Group Registration, 14
Service Invocation, 11
Service Processes, 10
Service Transport, 11
Service-Anfrage, 15
SetResourceProperties, 17, 58
setTerminationTime, 13, 58
Shibboleth, 37
Simple Object Access Protocol (SOAP), 9
Simple ResourcePropertySet, 60
Stub-Klassen, 67

TerminationTime, 16

UpdateResourceProperties, 18

Value metadata, 15
Verwaltung von Mitgliedern, 27
VO Auflösung, 27
VO Bildung, 24
VO Definition, 1
VO Dynamik, 2, 23
VO Management-Ansätze, 4
VO Membership Registration Service (VOMRS), 36
VO Membership Service (VOMS), 36
VO ResourceProperties, 48
VO Update, 27
VO-Anwendungsfälle, 23
VOFactoryService, 51
VOMemberPermit, 62
VOMS-Server, 36
VOResource, 58
VOResourceHome, 61

Web Service, 5, 6, 8, 15
Web Services Deployment Descriptor (WSDD), 67, 68
Web Services Description Language (WSDL), 15
Web Services Resource Framework (WSRF), 5
WS-Addressing, 14
WS-BaseFaults, 14, 43
WS-Invocation, 11
WS-Nachricht, 14
WS-Notification, 15
WS-Request, 11
WS-Resource, 15
WS-ResourceLifetime, 13
WS-ResourceProperties, 13
WS-ServiceGroup, 14
wsdlFile, 70
WSRF-Methode, 67